LEAPS League of European Accelerator-based Photon Sources

European HDF User Group summit
GPU processing of HDF5 data

Jérôme Kieffer
ESRF

Introduction

- Profile of a typical data reduction workflow
- Efficient multi-threading in Python ?
- Offloading processing to GPU
- And what about compression ?

Introduction

- Profile of a typical data reduction workflow
- Efficient multi-threading in Python ?
- Offloading processing to GPU
- And what about compression ?

Credit :
Marco Cammarata
ESRF ID18

Profile of a reduction workflow

http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Profile of a reduction workflow

http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

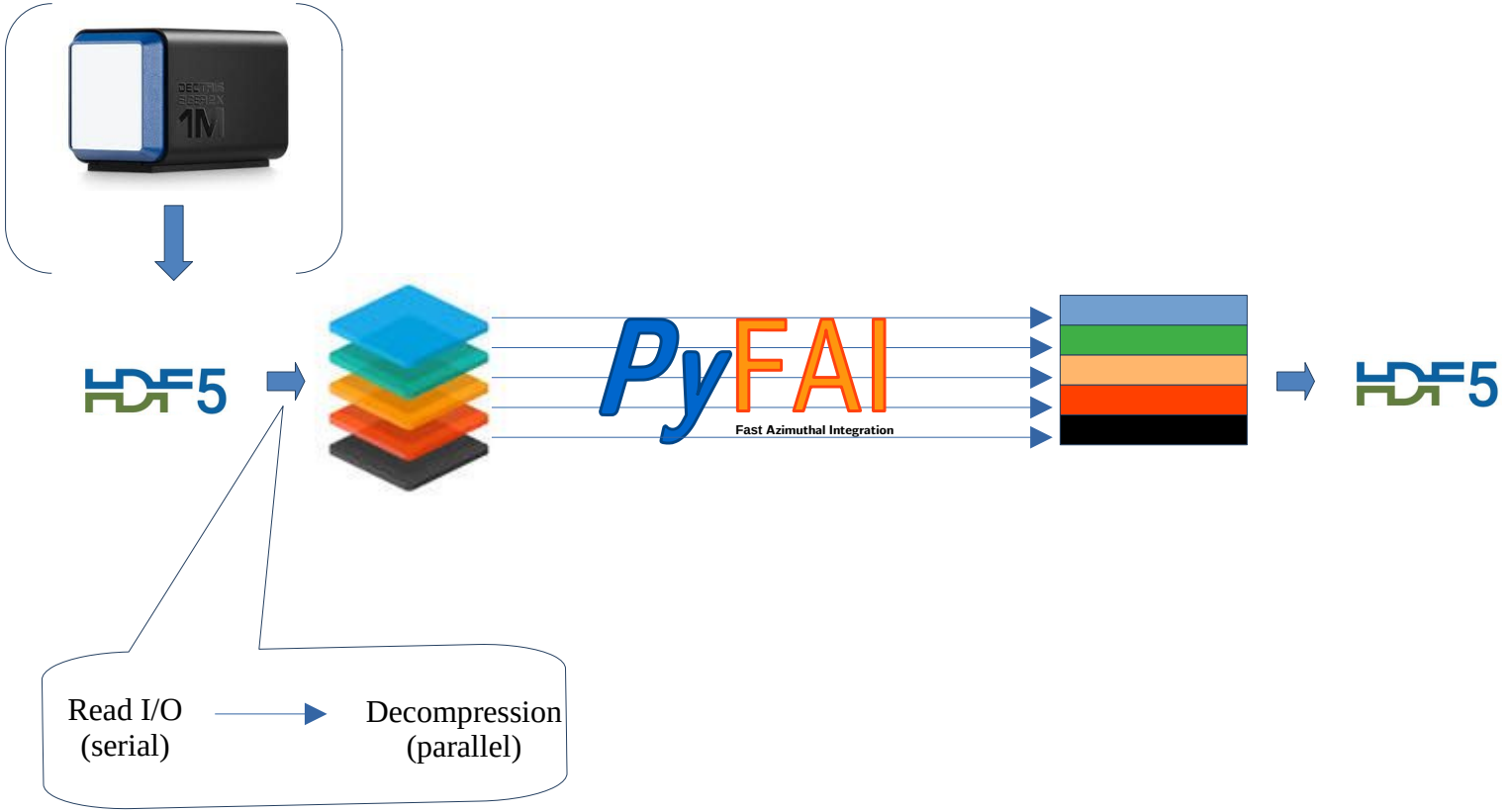
Profile of a reduction workflow

http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Profile of a reduction workflow

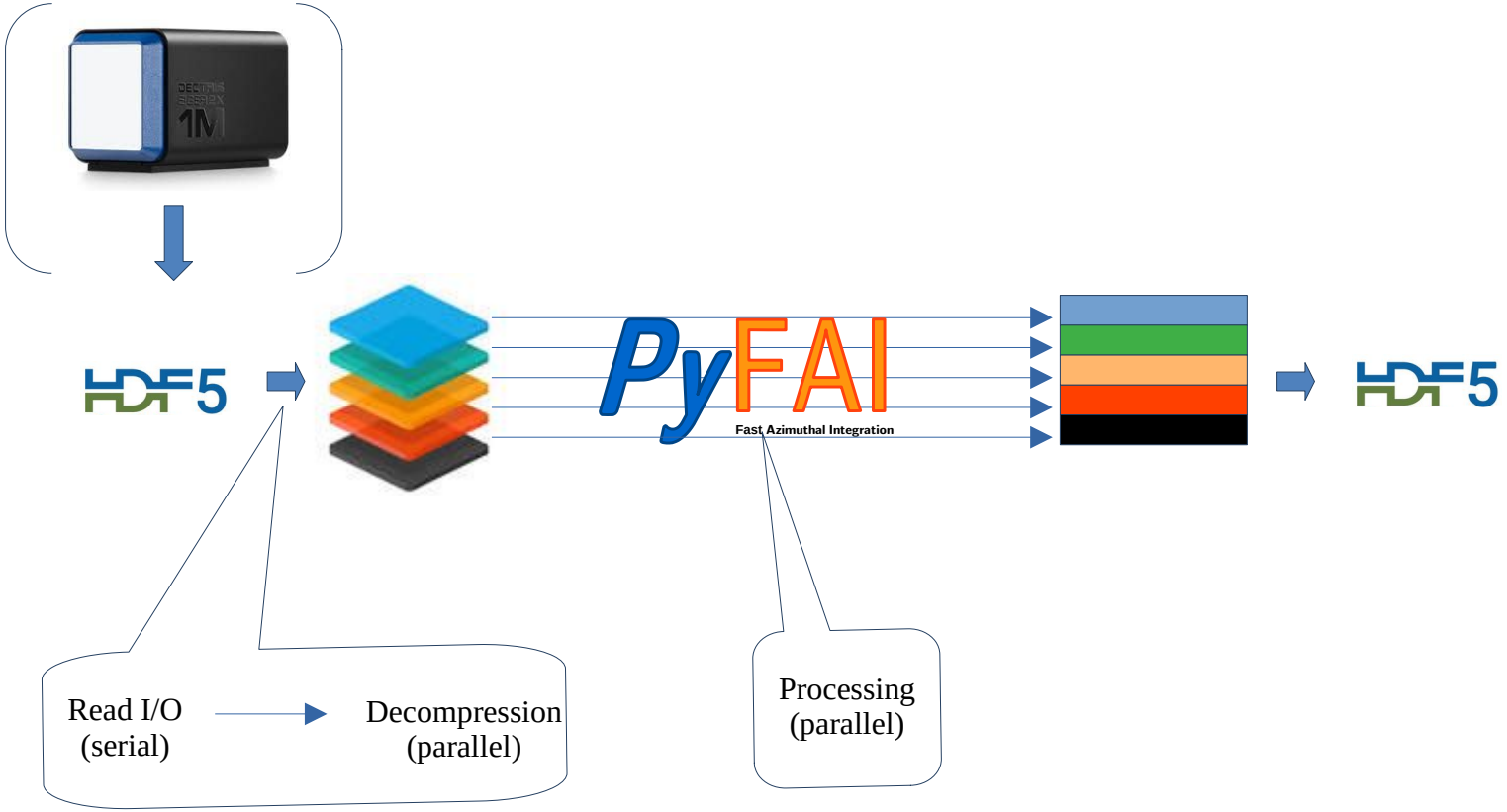
http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Profile of a reduction workflow



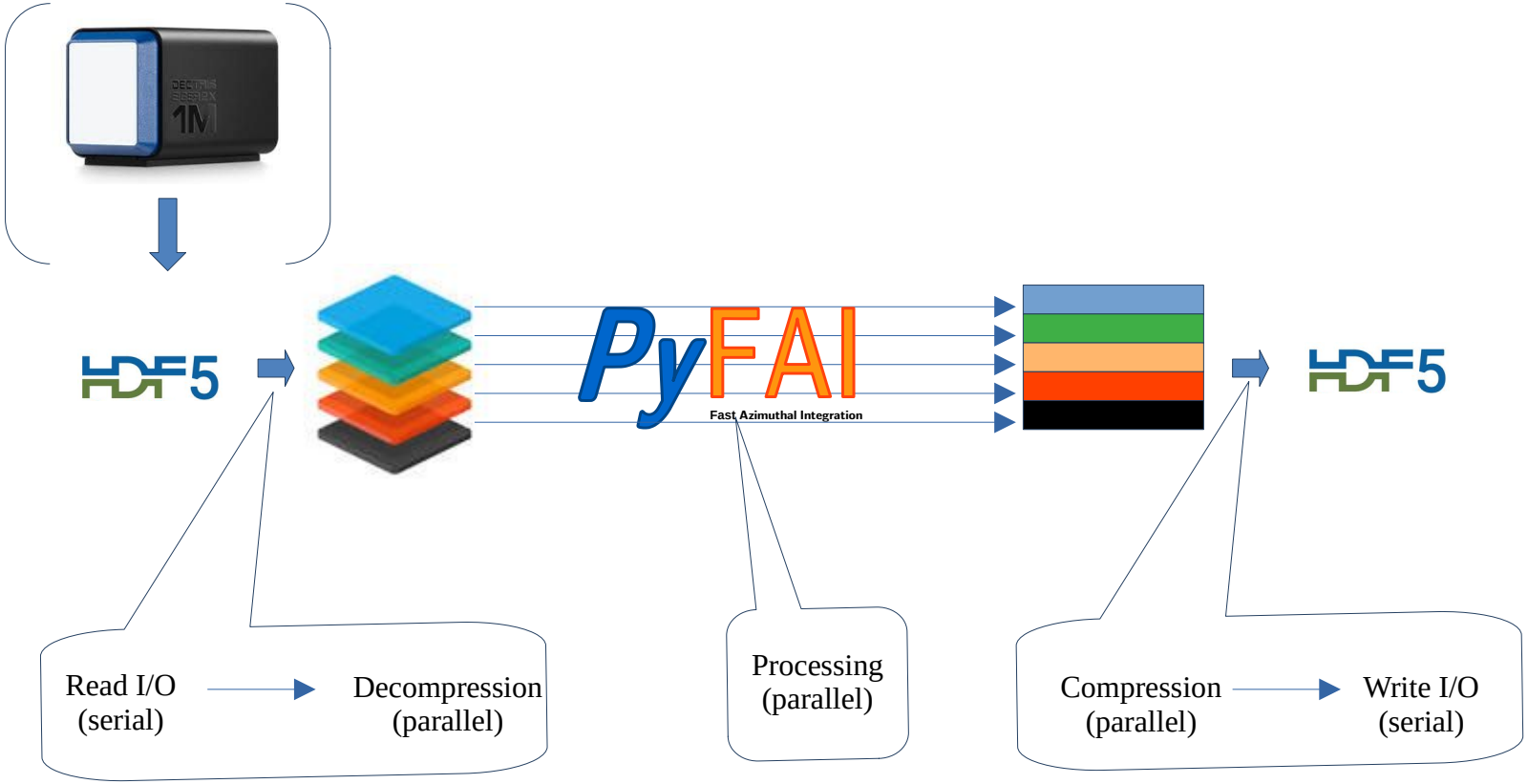
http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Profile of a reduction workflow



http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Profile of a reduction workflow



http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/index.html

Actual profiling from Python (h5py)

Setup:

- Computer: 2 x 32-core AMD EPYC 75F3
- Images: Eiger 4M (2070x2167, uint32), 4000 images
- Storage format: HDF5 bitshuffle-LZ4
- Reduction: 1D azimuthal integration 1000 bins (CSR, full split)

Actual profiling from Python (h5py)

Setup:

- Computer: 2 x 32-core AMD EPYC 75F3
- Images: Eiger 4M (2070x2167, uint32), 4000 images
- Storage format: HDF5 bitshuffle-LZ4
- Reduction: 1D azimuthal integration 1000 bins (CSR, full split)

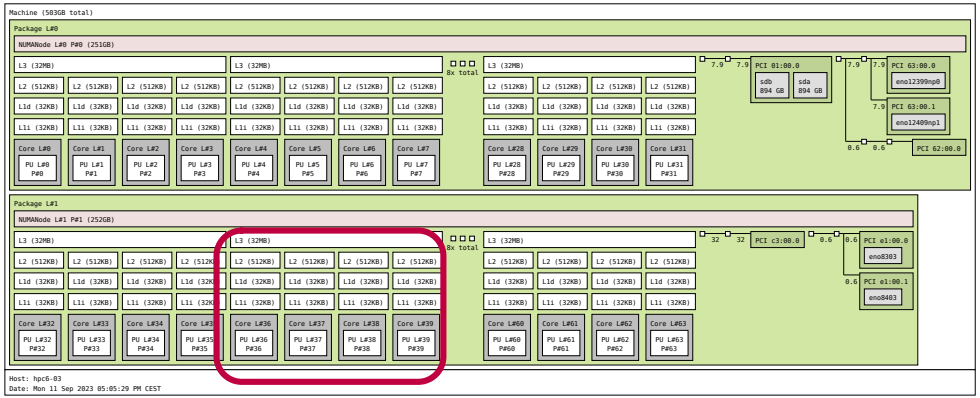
Performances	Read	Process	Read+Process
Serial (1t)	184 fps	25 fps	14 fps
Parallel (64t)	174 fps	5 fps	5 fps

Analysis

- Computer:
  - 2 NUMA-nodes
  - Slow communication
  - Much worse on AMD than Intel processors

Analysis

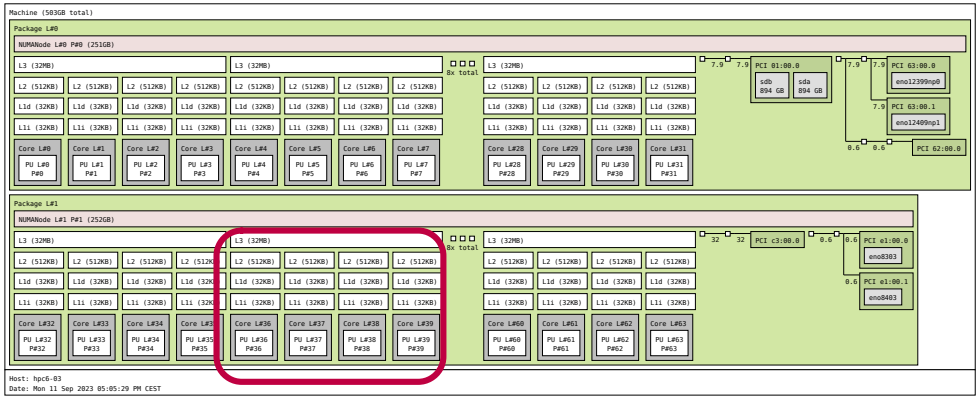
- Computer:
  - 2 NUMA-nodes
  - Slow communication
  - Much worse on AMD than Intel processors



LEAPS League of European Accelerator-based Photon Sources

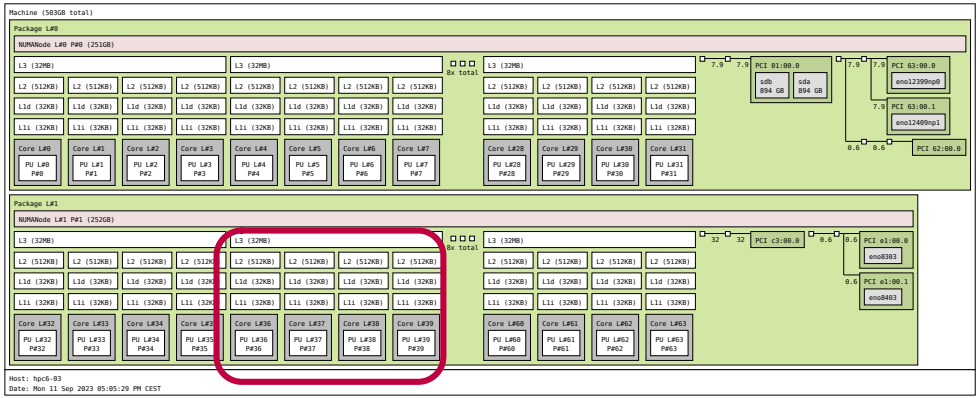
Analysis

- Computer:
  - 2 NUMA-nodes
  - Slow communication
  - Much worse on AMD than Intel processors
- Work around:
  - Process on a group of 4 cores sharing the same L3-cache



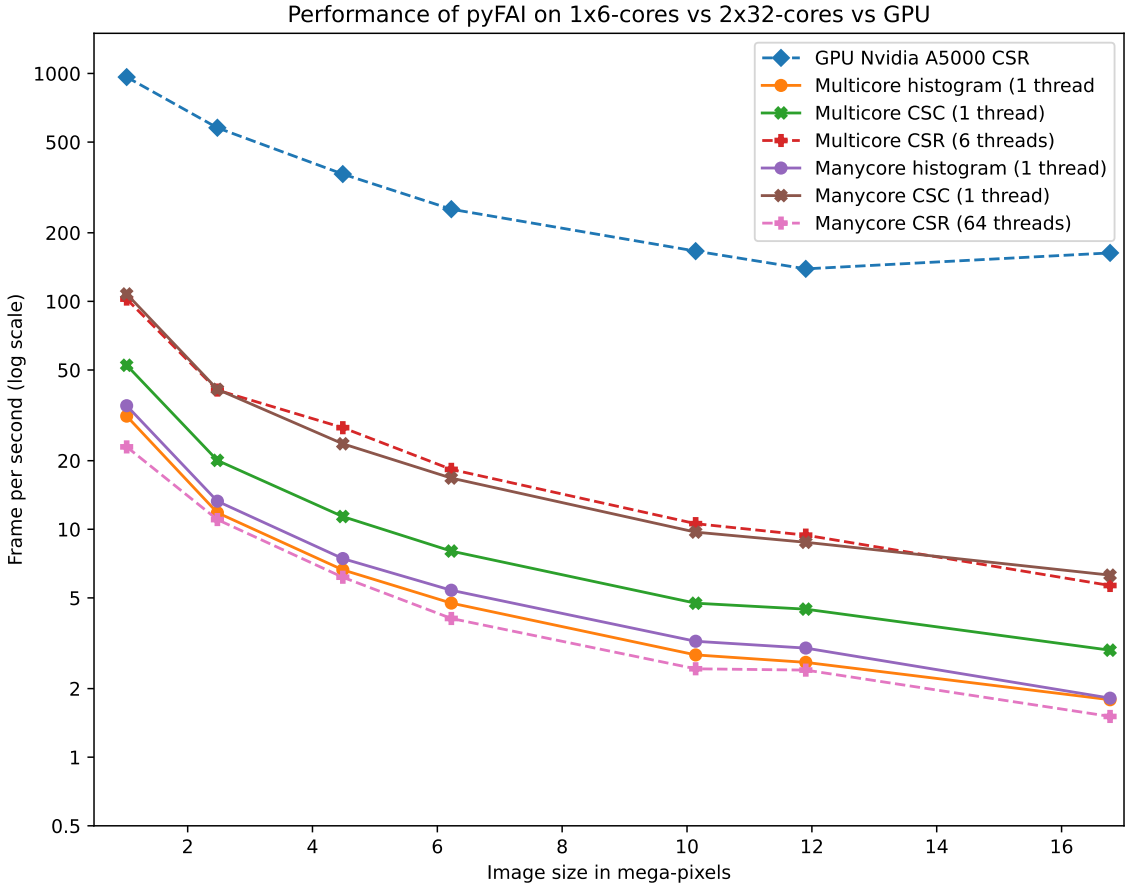
Analysis

- Computer:
  - 2 NUMA-nodes
  - Slow communication
  - Much worse on AMD than Intel processors
- Work around:
  - Process on a group of 4 cores sharing the same L3-cache



Performances	Read	Process	Read+Process
Serial (1t)	184 fps	25 fps	14 fps
Parallel (4t)	471 fps	21 fps	20 fps
Parallel (64t)	174 fps	5 fps	5 fps

Comparison manycore vs multicore



Manycore: Dual AMD EPYC 75F3, 64 cores (2021)
Multicore: Single Intel Xeon E5-1650 v4, 6 cores (2016)

Issues spotted:

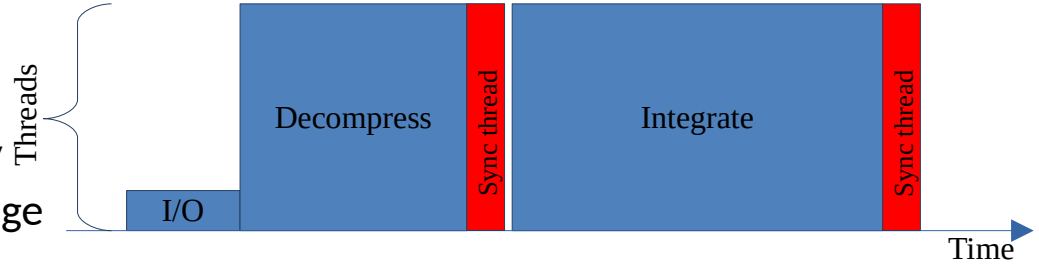
- Manycore computers do not play nicely with OpenMP
  - Memory transfer between sockets is a blocking factor.

Issues spotted:

- Manycore computers do not play nicely with OpenMP
  - Memory transfer between sockets is a blocking factor.
  - There are 2 threads synchronization per image to treat:
    - After reading
    - After processing
    - Cache inefficiency
      - Low CPU usage

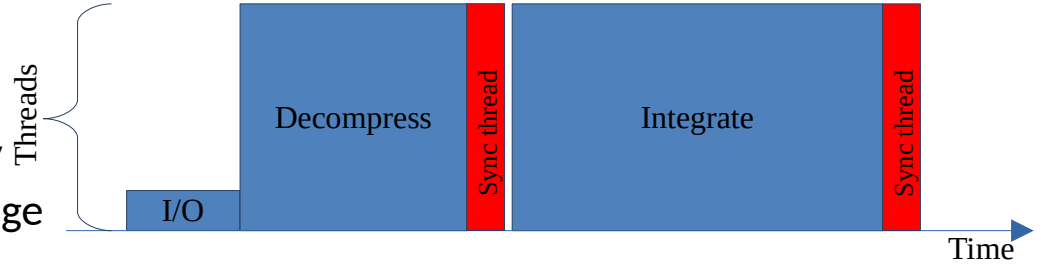
Issues spotted:

- Manycore computers do not play nicely with OpenMP
  - Memory transfer between sockets is a blocking factor.
  - There are 2 threads synchronization per image to treat:
    - After reading
    - After processing
    - Cache inefficiency
      - Low CPU usage



Issues spotted:

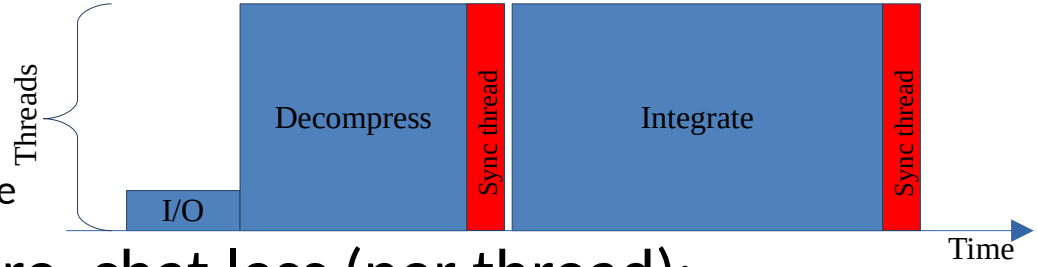
- Manycore computers do not play nicely with OpenMP
  - Memory transfer between sockets is a blocking factor.
  - There are 2 threads synchronization per image to treat:
    - After reading
    - After processing
    - Cache inefficiency
      - Low CPU usage



- Solution: work more, chat less (per thread):
  - One thread processes one or several images at once
    - Disable OpenMP in decompression algorithm (default in *hdf5plugin now)*
    - Decompress data and integrate image within the same thread
    - Optimized integration algorithm for single thread: CSC vs CSR sparse matrix

Issues spotted:

- Manycore computers do not play nicely with OpenMP
  - Memory transfer between sockets is a blocking factor.
  - There are 2 threads synchronization per image to treat:
    - After reading
    - After processing
    - Cache inefficiency
      - Low CPU usage



- Solution: work more, chat less (per thread):
  - One thread processes one or several images at once
    - Disable OpenMP in decompression algorithm (default in *hdf5plugin now)*
    - Decompress data and integrate image within the same thread
    - Optimized integration algorithm for single thread: CSC vs CSR sparse matrix
- Issue: Python Global Interpreter Lock

Piece-wise profiling (**timeit**):

- Reading (pure I/O):
  - Use **h5py.Dataset.id.read_direct_chunk**
  - Read speed: 3200 fps

Piece-wise profiling (**timeit**):

- Reading (pure I/O):
  - Use **h5py.Dataset.id.read_direct_chunk**
  - Read speed: 3200 fps
- Decompression on a single core:
  - Use **bitshuffle.decompress_lz4**
  - Decompression speed: 244 fps

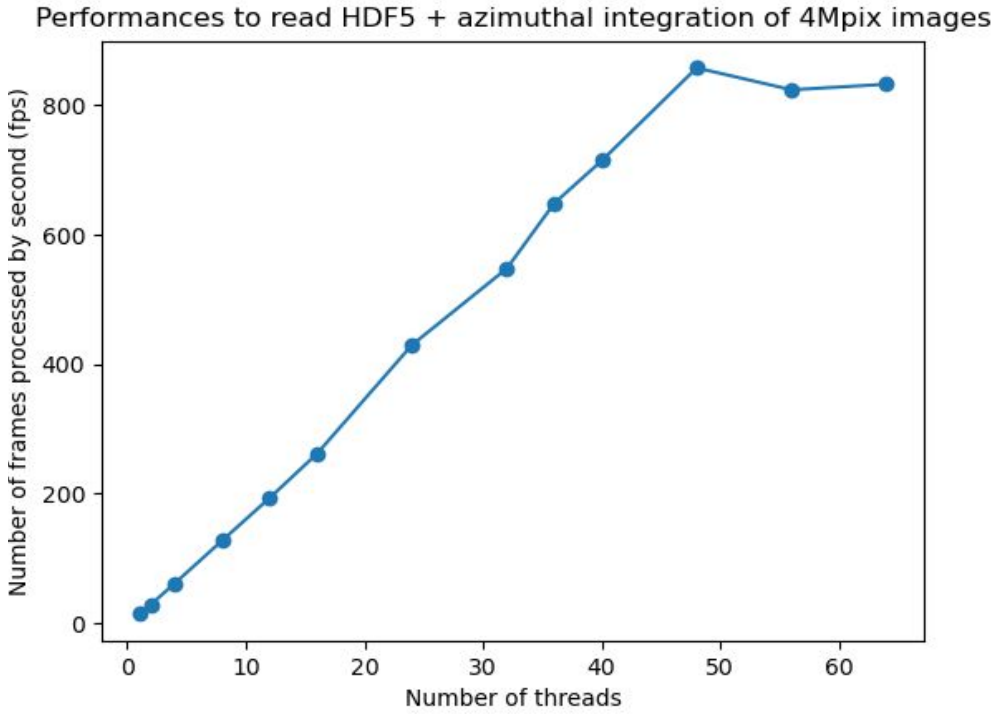
Piece-wise profiling (**timeit**):

- Reading (pure I/O):
  - Use **h5py.Dataset.id.read_direct_chunk**
  - Read speed: 3200 fps
- Decompression on a single core:
  - Use **bitshuffle.decompress_lz4**
  - Decompression speed: 244 fps
- Integration on a single core:
  - Using **CSR** sparse matrix: 15 fps
  - Using **CSC** sparse matrix: 25 fps

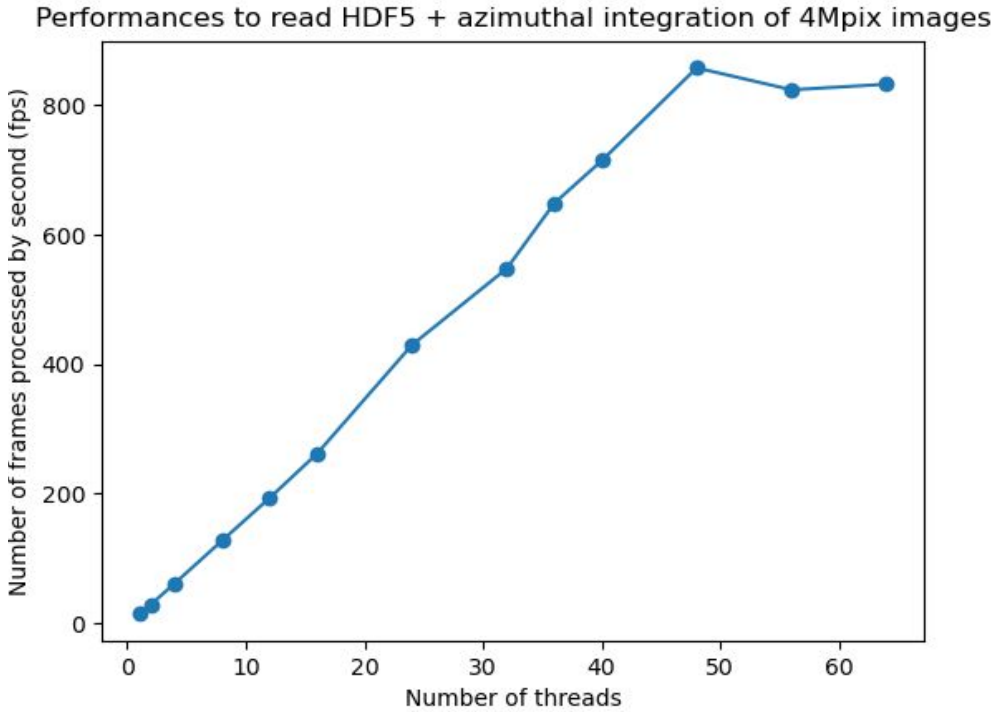
Piece-wise profiling (**timeit**):

- Reading (pure I/O):
  - Use **h5py.Dataset.id.read_direct_chunk**
  - Read speed: 3200 fps
- Decompression on a single core:
  - Use **bitshuffle.decompress_lz4**
  - Decompression speed: 244 fps
- Integration on a single core:
  - Using **CSR** sparse matrix: 15 fps
  - Using **CSC** sparse matrix: 25 fps
- All together:
  - In a simple python function: 22 fps

Use Python threads ! (No TROLL)



Use Python threads ! (No TROLL)



- Works efficiently when most code is GIL-free like:
  - Bitshuffle-LZ4 decompression
  - Sparse matrix multiplications

Conclusion: Multi-threading

- Many-cores do not behave nicely with OpenMP
  - Visible on multi-socket system and all AMD EPYC (NUMA)
  - Hdf5plugin has now disabled OpenMP by default !
  - Performances can be much worse than no parallelization
- A pool of thread outperforms OpenMP
  - Requires more code (100 loc instead of 10)
  - Scales linearly up to ~40 cores (22 → 850fps).
- Multi-processing is also an option
  - Serialization issue

What about GPUs ?

GPU data reduction

- Processing:
  - PyFAI runs on GPU since 2012 (published 2014)
    - Implemented in OpenCL
      - Tested on CPU & GPU Nvidia, Intel, AMD, …
    - https://arxiv.org/abs/1412.6367

GPU data reduction

- Processing:
  - PyFAI runs on GPU since 2012 (published 2014)
    - Implemented in OpenCL
      - Tested on CPU & GPU Nvidia, Intel, AMD, …
    - https://arxiv.org/abs/1412.6367
- Decompression:
  - Nvidia
    - 2020: *nvcomp* started as open source project
    - 2022: has been made closed source
    - And of course: CUDA only

GPU data reduction

- Processing:
  - PyFAI runs on GPU since 2012 (published 2014)
    - Implemented in OpenCL
      - Tested on CPU & GPU Nvidia, Intel, AMD, …
    - https://arxiv.org/abs/1412.6367
- Decompression:
  - Nvidia
    - 2020: *nvcomp* started as open source project
    - 2022: has been made closed source
    - And of course: CUDA only

Credit : Jon Wright, ESRF ID11

GPU data reduction

- Processing:
  - PyFAI runs on GPU since 2012 (published 2014)
    - Implemented in OpenCL
      - Tested on CPU & GPU Nvidia, Intel, AMD, …
    - https://arxiv.org/abs/1412.6367
- Decompression:
  - Nvidia
    - 2020: *nvcomp* started as open source project
    - 2022: has been made closed source
    - And of course: CUDA only
  - OpenCL
    - 2022: silx implements a LZ4 decompressor and bitshuffling
    - 2023: compressor work on LZ4 ongoing

Credit : Jon Wright, ESRF ID11

Azimuthal integration on GPU:

- Histograms
  - based on atomic addition (limited performances)

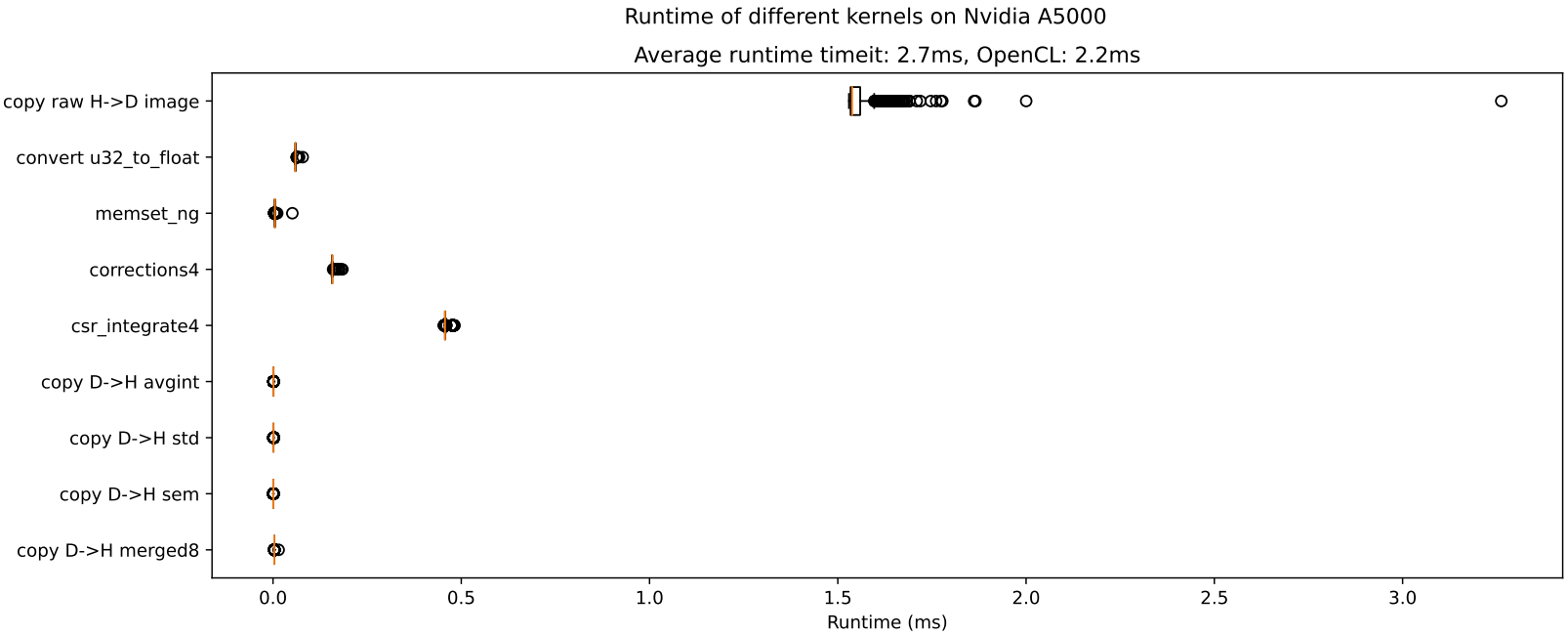
Azimuthal integration on GPU:

- Histograms
  - based on atomic addition (limited performances)
- CSR sparse matrix dense vector multiplication
  - Well suited to GPU (→1500 fps, limited by PCIe)
  - Implements error propagation based on
    - Provided variance or poissonian error model
    - Variance extracted from the ring's average

Azimuthal integration on GPU:

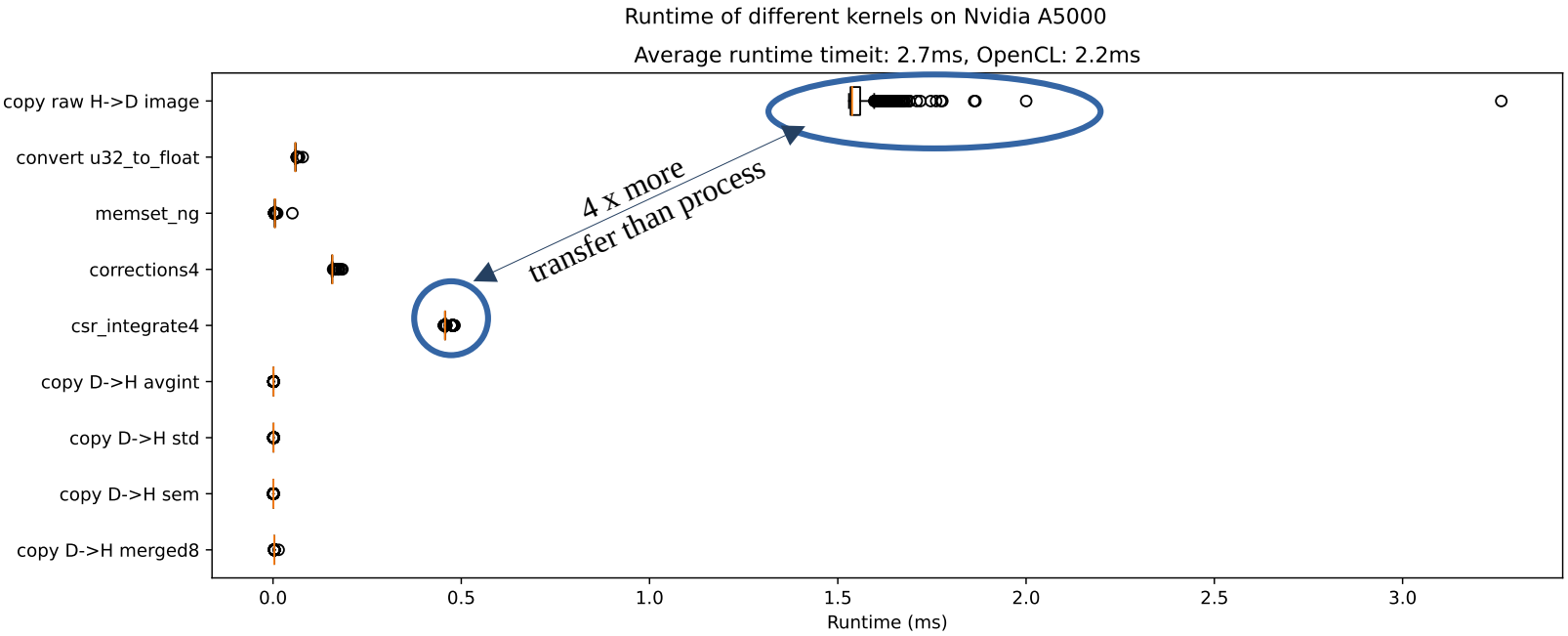
- Histograms
  - based on atomic addition (limited performances)
- CSR sparse matrix dense vector multiplication
  - Well suited to GPU (→1500 fps, limited by PCIe)
  - Implements error propagation based on
    - Provided variance or poissonian error model
    - Variance extracted from the ring's average
- Double precision emulated using double-float
  - http://www.theses.fr/2017LYSEN036

Profiling of GPU azimuthal integration



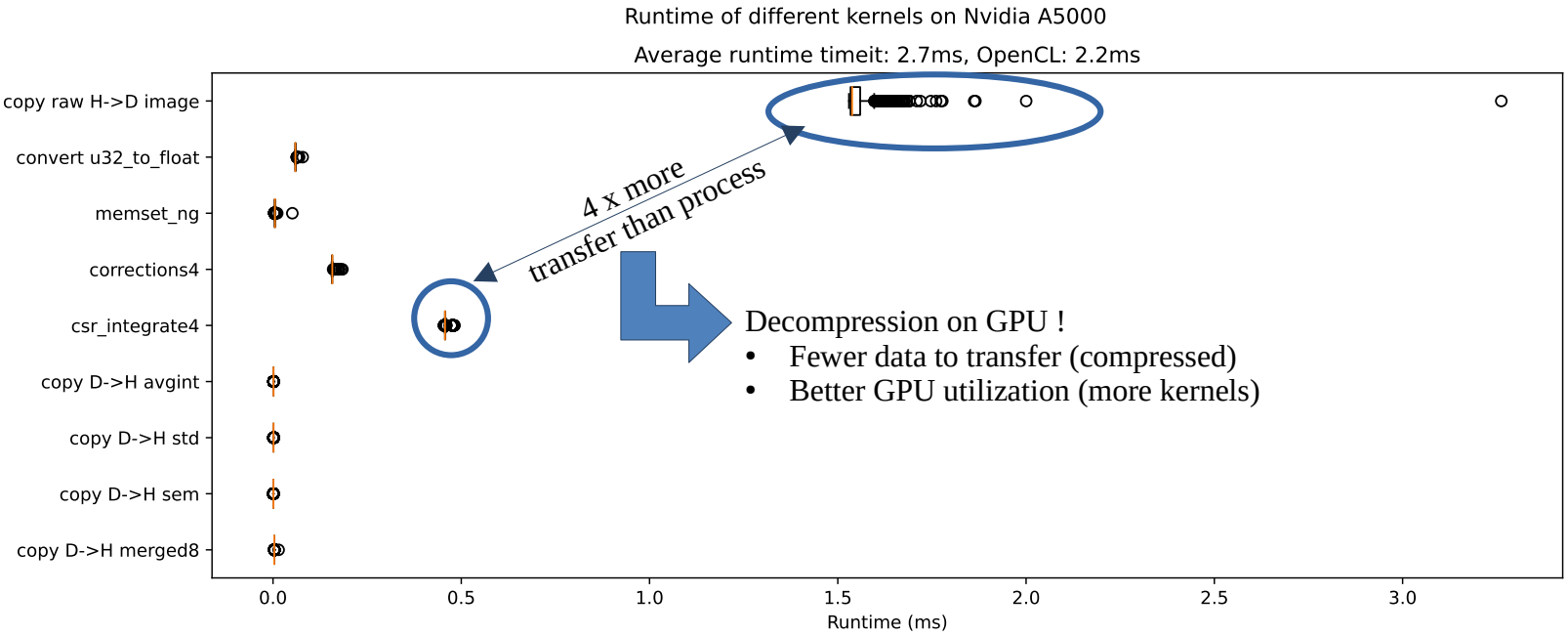
- Parameters:
  - Integration of Eiger 4M images
  - Computer: Xeon E5 1650v4 + Nvidia A5000 (PCIe v3)

Profiling of GPU azimuthal integration



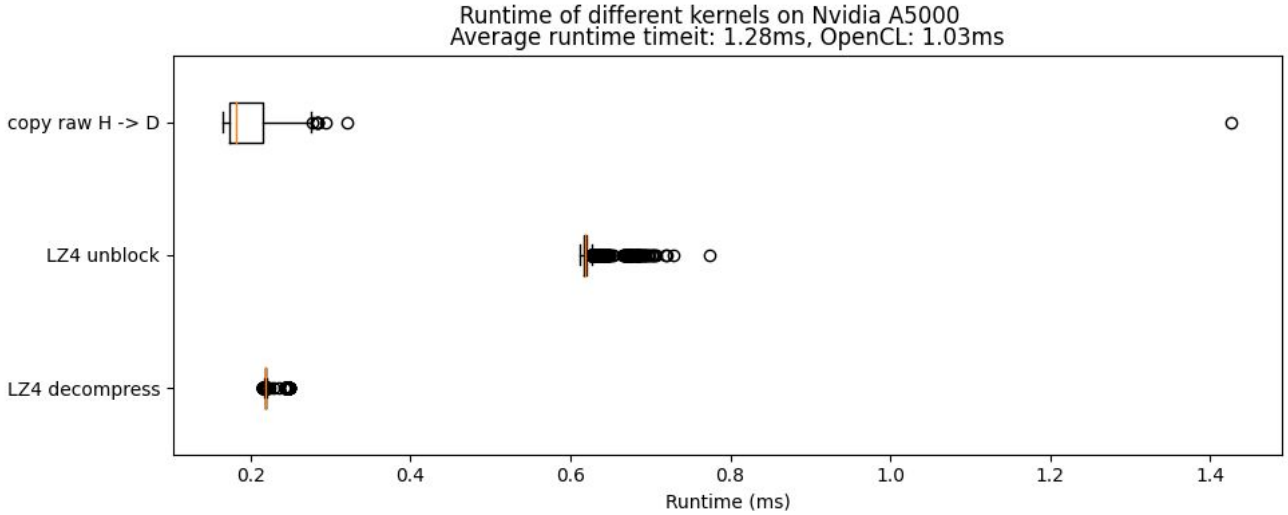
- Parameters:
  - Integration of Eiger 4M images
  - Computer: Xeon E5 1650v4 + Nvidia A5000 (PCIe v3)

Profiling of GPU azimuthal integration



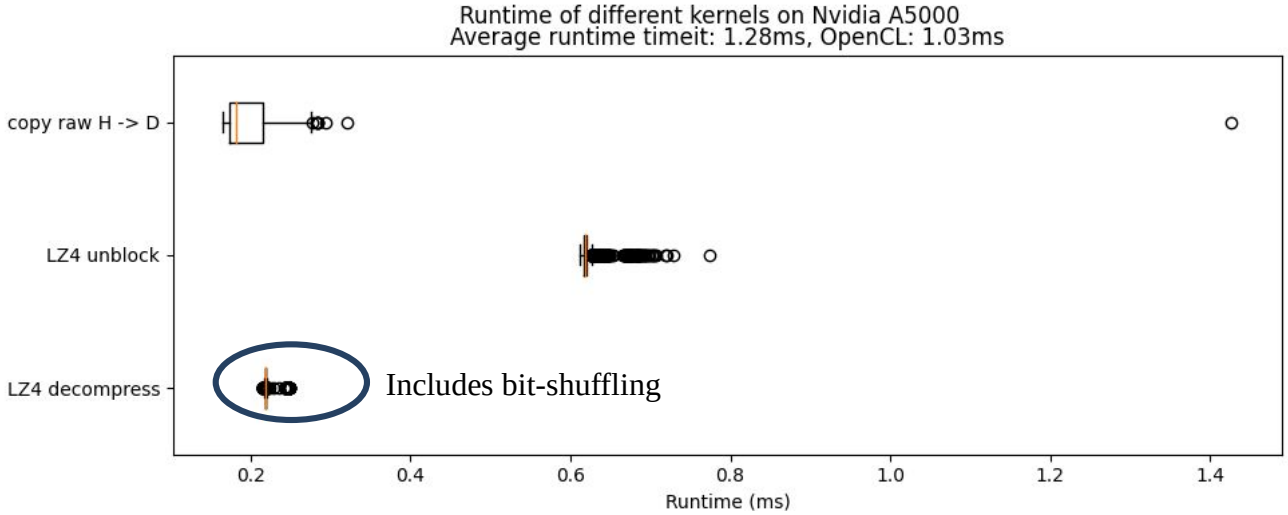
- Parameters:
  - Integration of Eiger 4M images
  - Computer: Xeon E5 1650v4 + Nvidia A5000 (PCIe v3)

Profiling LZ4 decompression on GPU

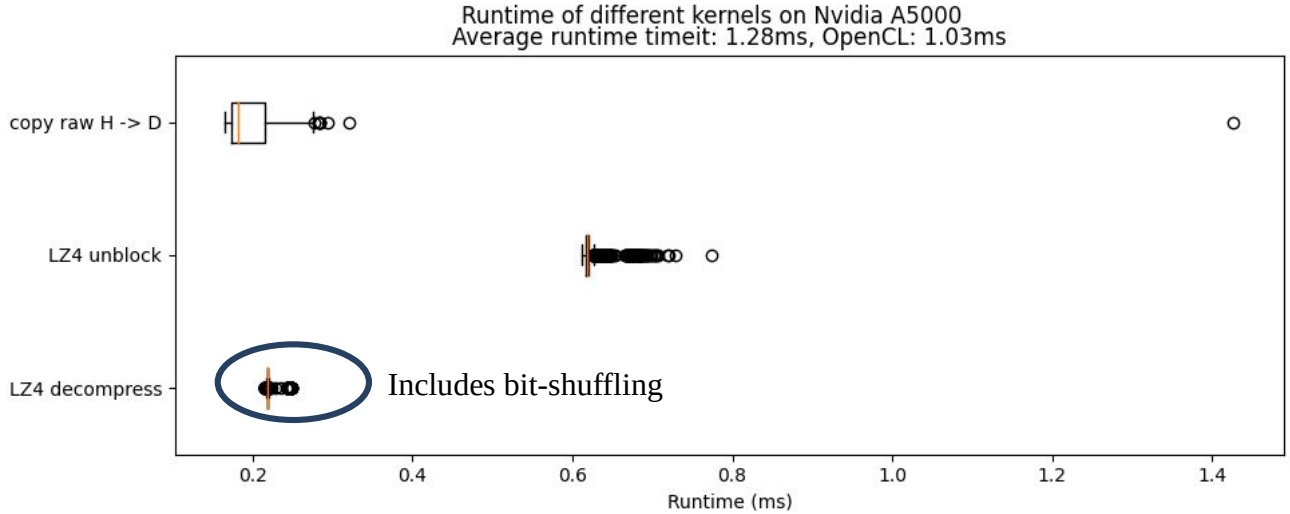


LEAPS League of European Accelerator-based Photon Sources

Profiling LZ4 decompression on GPU

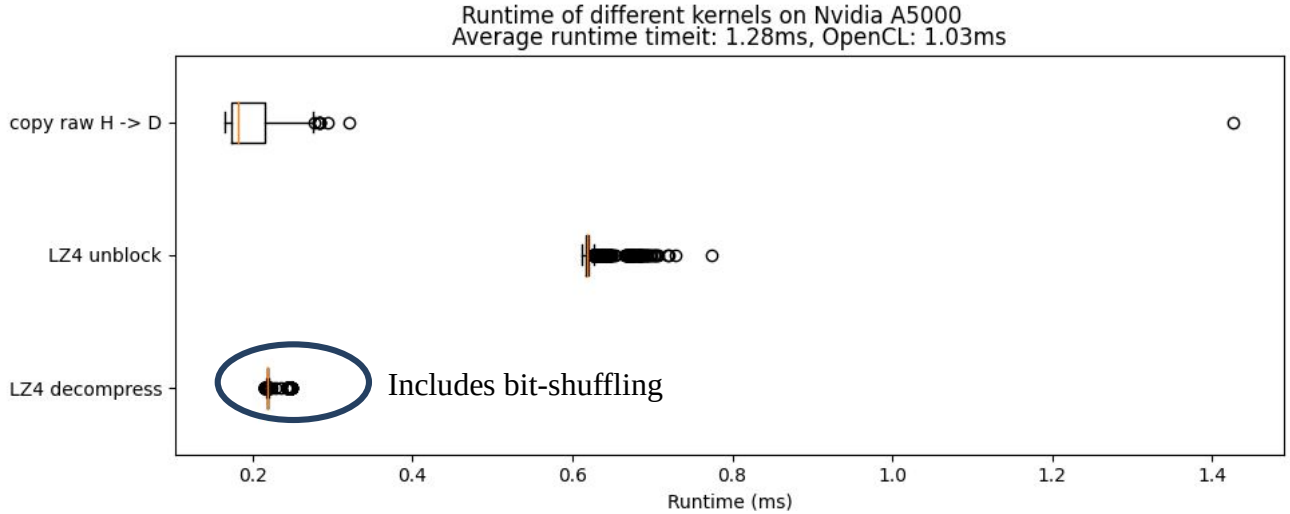


Profiling LZ4 decompression on GPU



- LZ4 was designed for in-cache CPU workload:
  - → 974 fps on a Xeon (6-cores)

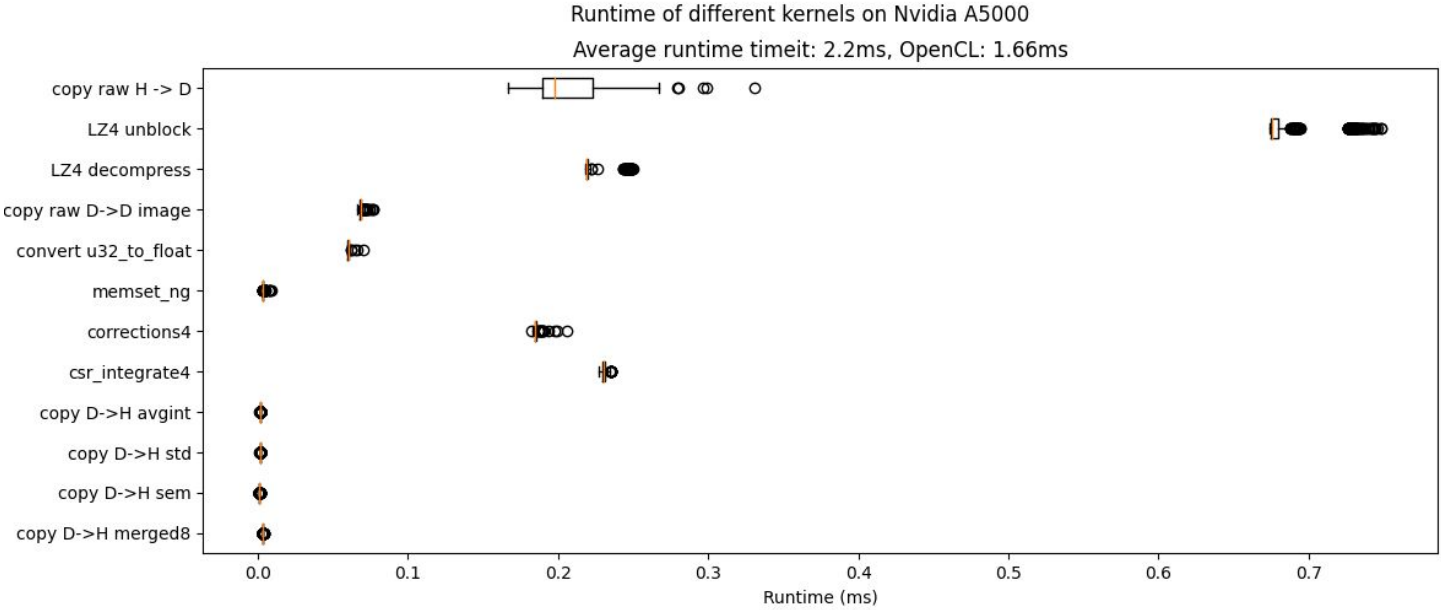
Profiling LZ4 decompression on GPU



- LZ4 was designed for in-cache CPU workload:
  - → 974 fps on a Xeon (6-cores)
- Implementation by Nvidia finds blocks on CPU
- Implementation from silx finds segments on GPU:
  - →887 fps + extra savings from fewer data to send to the GPU

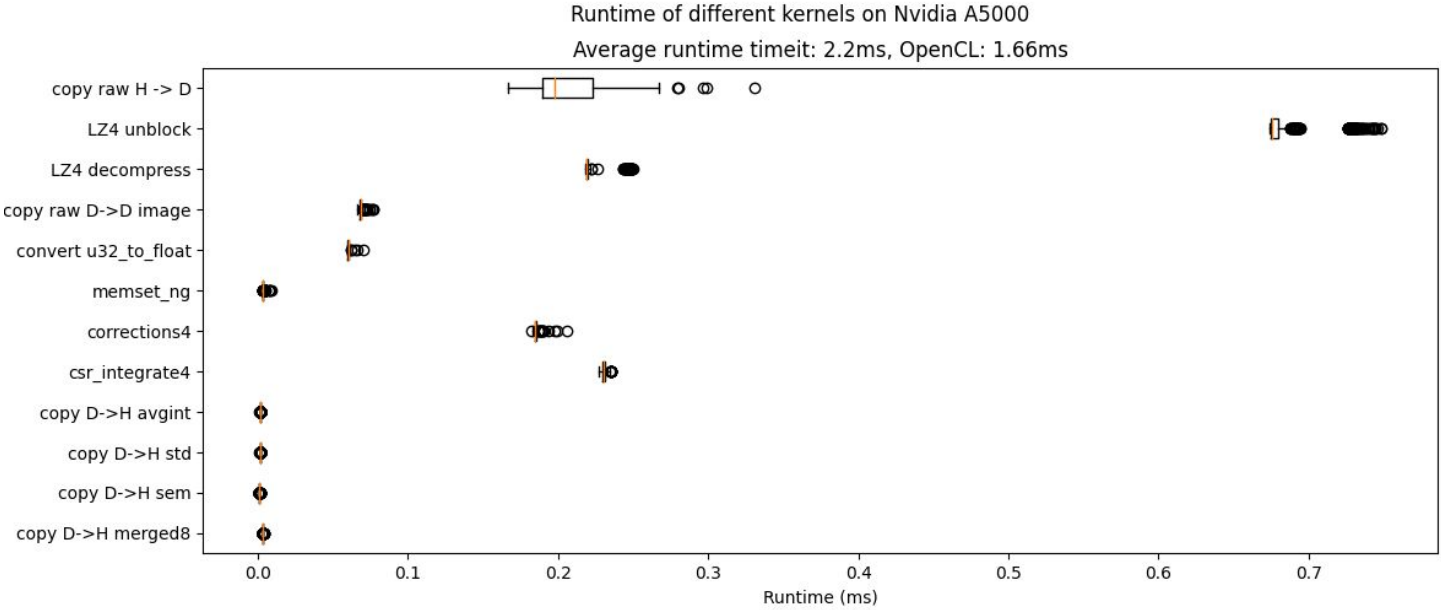
Profiling the complete pipeline

- Transfer → decompression → integration



Profiling the complete pipeline

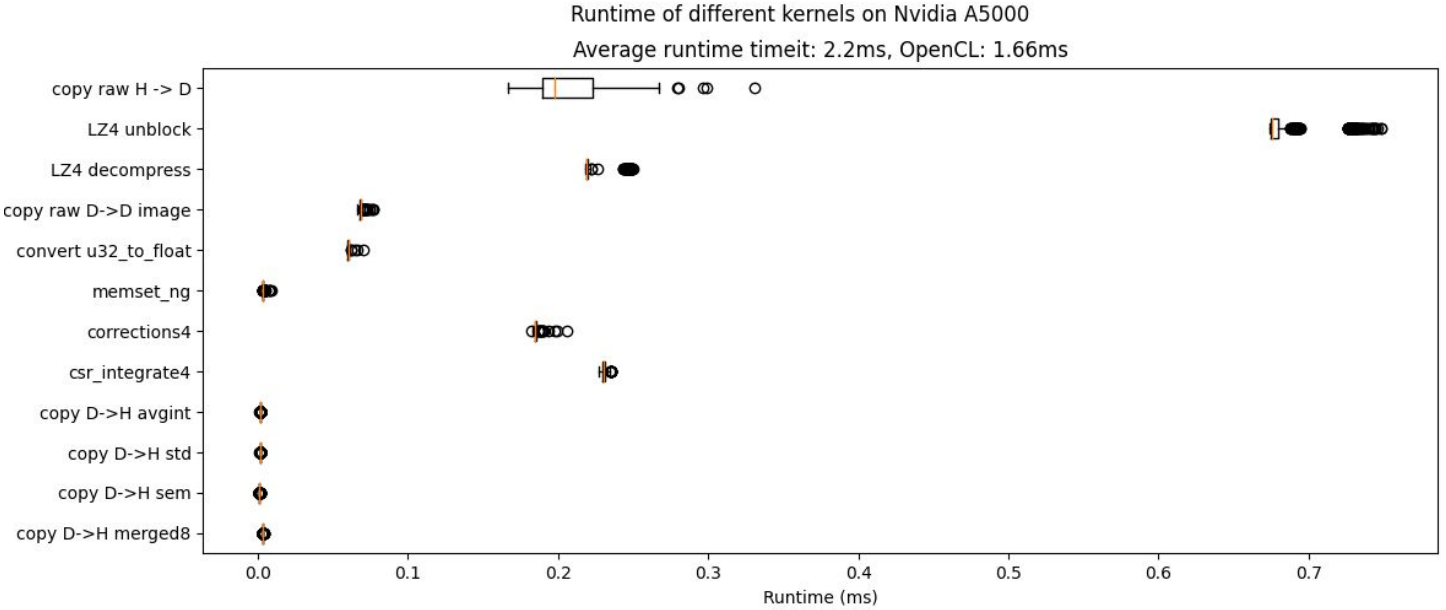
- Transfer → decompression → integration



- Performances: 450 fps (single process/thread)

Profiling the complete pipeline

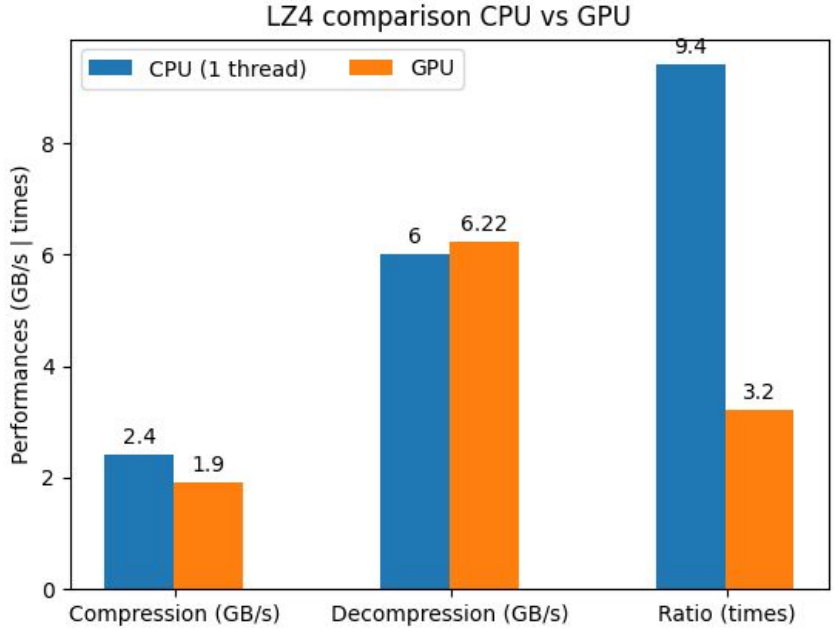
- Transfer → decompression → integration



- Performances: 450 fps (single process/thread)
- Multiprocessing is possible: 568 fps with 2 workers
  - http://www.silx.org/doc/pyFAI/dev/usage/tutorial/Parallelization/MultiGPU.html

A word on LZ4 compression

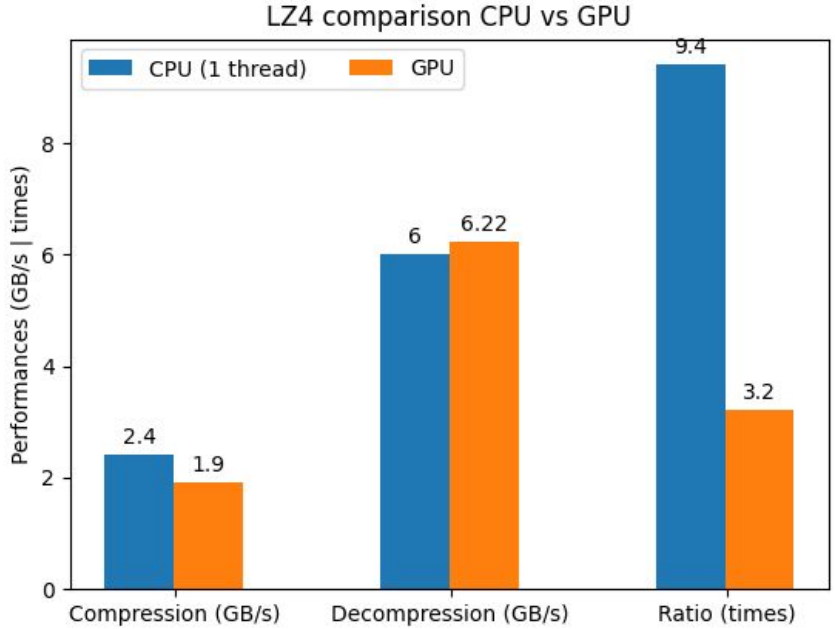
- Using *nvcomp* provided by Nvidia (closed source)



Test data have been bit-shuffled before hand

A word on LZ4 compression

- Using *nvcomp* provided by Nvidia (closed source)



Test data have been bit-shuffled before hand

- LZ4 algorithm is not well suited to GPU !
  - Thanks to bit-shuffling saves in BW compensates speed

Final word on energy consumption

- Workstation (from 2016)
  - Xeon E5 1650 v4 processor: 70W (measured)
  - OpenMP CSR algorithm: 35 fps
    - → **2 J/frame** (not accounting the decompression)

Final word on energy consumption

- Workstation (from 2016)
  - Xeon E5 1650 v4 processor: 70W (measured)
  - OpenMP CSR algorithm: 35 fps
    - → **2 J/frame** (not accounting the decompression)
- HPC server (from 2021):
  - Dual AMD Epyc 75F3 (280W each, estimated from TDP)
  - Multi-threaded: 850 fps
    - → **0.7 J/frame**

Final word on energy consumption

- Workstation (from 2016)
  - Xeon E5 1650 v4 processor: 70W (measured)
  - OpenMP CSR algorithm: 35 fps
    - → **2 J/frame** (not accounting the decompression)
- HPC server (from 2021):
  - Dual AMD Epyc 75F3 (280W each, estimated from TDP)
  - Multi-threaded: 850 fps
    - → **0.7 J/frame**
- GPU in a workstation
  - NVIDIA RTX A5000: 215W (145W measured + 70W for the system)
  - GPU implementation: 450 fps (2 processes: 568 fps)
    - → **0.5J/frame** (0.4 J/frame with 2 processes)

Thanks to ...

- Inspiring scientists:
  - Marco Cammarata
  - Jon Wright
- Helpful colleagues:
  - Thomas Vincent
  - Armando sole (hdf5plugin)
  - Pierre Paleo
  - Alejandro Homs
  - Samuel Debionne
- Open source community:
  - HDF5
  - h5py