Efficient use of HDF5 with high data rate x-ray detectors – Introduction

Some notes on the background or general conditions for the HDF5 workshop at PSI on May 30.-31., 2012. I did not circulate this document for revision, so it's my view only.

Heiner Billich, Heiner.Billich@psi.ch, May 2012

Set the (future) scene

Users take data at an experimental station or *beamline*. Regarding *data volume* almost all data origins from a 2d Eiger pixel detector - sequences of images or *frames*.

Single frames have 100K-16M pixels and a single unsigned integer value per pixel. All frames in a single sequence, scan or experiment have the same number pixels and the same precision (number of bits) per pixel. Hence the detector data for a single scan or experiment maps naturally to a simple array with rank=3 and integer datatype. The array's dimensions are (image-number, y, x) with detector pixel coordinates (y,x)

Users take data at sustained uncompressed rates of

- up to 5GByte/s with commercial Dectris Eiger detectors and
- several 10GByte/s with research versions of Eiger maintained at PSI.

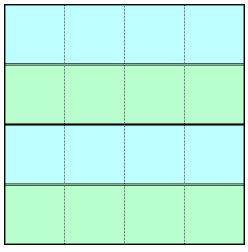
Single experimental datasets hold a few thousand to a few tens of thousands of images. Extreme cases may result in datasets with millions of images.

In parallel to the data acquisition users

- run data analysis jobs on a compute cluster or beamline workstations
- run preview and quality control tools to monitor the experimental data
- transfer the data to removable media or via network to their home place

Each beamline has some dedicated space on permanent disk storage. This *beamline storage* is available through standard NFS or CIFS protocols and may allow native access to an underlying parallel file system like GPFS or Lustre, too.

The Eiger Detector



Eiger is a modular 2d single photon counting x-ray detector. The basic mechanical building block of all Eiger detectors is a rectangular ½ M pixel unit or *module* with 512x1024 active pixels. Several tiled *modules* form detectors with larger pixel counts: 1M, 4M, 9M, 16M ,… .

The picture on the lefts shows the geometrical structure of a 1M Eiger: The four square chips in each row form a ¼ M pixel *half-module*. Two half modules add up to a ½ M pixel Eiger module and two modules form the full 1M detector.
A full detector image or frame is the sum of the individual module's *sub-images*. For the example 1M detector the full 1M pixel image holds two ½ M module sub-images or four ¼ M half-module sub-images.

Eiger will come in two flavors – a commercial version from Dectris and a research version maintained by PSI:

Research version of Eiger - scale-out to extreme data rates

Each half-module of ¼ M active pixels has it's own dedicated 10GbE data port, a 1GbE control port and dedicated electronics for detector readout, data buffering and initial data processing. Hence the available bandwidth for sustained pixel data flowing out of the detector scales linearly with the number of half modules: A single module with 500K pixles has two 10GbE data ports, a 1M detector build from two modules provides four 10GbE ports and a 9M has 36 individual 10GbE ports.
Larger versions of Eiger with 4, 16, 36 , … 10GbE ports can stream out pixel data at sustained rates of 4, 16, 36, … GB/s and scale-out in a perfect way, the achievable sustained data rates grow linearly with the number of detector modules.
The whole data flow from detector module to beamline storage will stay parallel and modular. The number of data–receiving detector-servers, network lines and file servers will increase with the number of detector modules in use, too. We can match the perfect scale-out of Eiger by adding servers, network paths and disk spindles as needed.

Commercial version of Eiger – scale-up but stay with a single detector server

Dectris' Eiger versions will all use a single detector-server with a bandwidth of ~5GB/s pixel data from the detector to the detector server, irrespective of the detector's pixel count. This more traditional layout fits better to a detector to be deployed at many different sites. Still 5GB/s is about eight to ten times the rates of current Pilatus detectors, Dectris' Eiger version does scale up the single detector-server setup.

Why compression, can't we just write the raw data?

Pixel detector data compresses well

Pixel detector data in general compresses well. For Pilatus we get compression by a factor of four with a simple byte-offset compression and 32bit integer data. The CBF file format used to store Pilatus data provides byte-offset compression. Gzip or bzip2 compress even better, factors of 10 or above are possible. For Eiger data we expect similar or better compression ratios.

Only byte-offset compression made Pilatus possible

Without compression and hence four-fold increased data rates and data volumes we couldn't even run the current Pilatus detectors at their native speed, beamline storage and network would limit the detector dramatically. When we introduced Pilatus at SLS compression probably was the most important factor which enabled high data-rate operation not limited by the beamline IT from the beginning.

Average sites can't handle 5GB/s sustained throughput

Only compression will enable sites with an average IT infrastructure to run the commercial Eiger detectors at the highest data rates: The raw data rates of ~5GB/s will reduce to below 1GB/s compressed data. 1 GB/s is about the maximum to expect from an average beamline IT infrastructure while 5GB/s and above will be available on a few large sites only. Actually today I know of no synchrotron beamline which can handle 5GB/s inflow from a single detector, even 1GB/s still is challenging.

Can't we use HDF5 just as it is?

Compression does not scale-up

You can't increase the achievable compression throughput to a single HDF5 file by adding more cores – compression filters are single threaded and HDF5 runs filters on one chunk of data at each given time only.

Compression does not scale-out if you want to stay with a reasonable number of files

You can't increase the compression throughput to a single HDF5 file by adding more nodes. If you're willing to split your data into many separate HDF5 files you need to write one file per processor core due to HDF5's limit that only a single thread can compress data. With an achievable compression throughput of well bellow 1GB/s/core with zlib we would even need to split a single half-module's data into several files and end up with over 50 files for a 9M detector and an overly complicated scattering of the detector data across many files.

Brute-force will not work

For the research version of Eiger with its scale-out design a brute-force approach without compression would technically probably work, but costs and required man-time to run such a large environment would scale-out linearly, too. Four-fold compression will result in four-times less infrastructure costs and may well make the difference to reach an affordable price for IT infrastructure. Actual Eiger data will likely compress by factors of 10 or above and make the difference even more pronounced.

Achievable throughput with HDF5 does not scale if compression is required

To sum up HDF5 does not allow us to scale throughput when we write compressed datasets. This is a major issue as we can't scale to the data rates required to operate Eiger detectors.

In an ideal world

Users take home a few HDF5 files …

… with compressed data

… with all metadata required to do further analysis

… files which they can open and read with plain HDF5 easily

At the beamline we …

… write compressed data at ~5GB/s incoming rate and < 1GB/s compressed on a single server

… can scale throughput to disk with the number of servers and cores, network ports and disk spindles

… can scale the overall compression throughput on writes by adding servers and cores

… write to the final HDF5 format, don't need to repack the data before users take it home

… never need to write uncompressed data to disk storage, compression throughput matches sustained detector output rates.

… achieve reasonable compression factors of 10 and above for typical pixel detector data

… write in parallel from several processes on several servers in an independent or asynchronous mode

The HDF5 files …

… allow to iterate through the images easily using the HDF5 API only

… allow close to maximum throughput if images are read sequentially one by one

… present the images as unsigned integer dataset with *rank=3*

… contain all metadata required for further processing and analysis

… adhere to some standard which allows to interpret the metadata in the file

… allow to detect data corruption

The experimental dataflow at the beamline …

… treats beamline storage as data sink primarily

… uses message passing to pass experimental data around

… reuses data in memory and does not re-read data unnecessarily from disk

… does large block sequential reads whenever it access experimental data on beamline storage

… does not unnecessarily multiply the number of files on disk with result files

Access patterns

The access pattern for writes at the beamline is very simple and limited. This may allow optimizations not possible for more general write patterns. Read patterns are simple, too. Once the file is written at the beamline all further accesses are read-only.

Write

Write once

Image or sub-image data is written once and never is overwritten.

Images written in sequential order

Image or sub-image data arrives in a sequential order. We can maintain this order up to the point where we write the data to file: We write data sequentially image by image or sub-image by sub-image.

Only whole images (or sub-images) written by single write() call

We write only whole images or sub-images, we never write partial images or sub-images. Neither do we write to any complex hyperslabs or selections. We may want to merge several consecutive image writes to a single write for performance reasons, but it's not required.

Fixed size of write requests

We write data with a fixed size for all write requests – either a full image or a fixed-size sub-images.

Exact write pattern know in advance

We know in advance the exact write pattern, which data-location we will write to by which process in which sequence. The only variation is in the timeline in a single process and the timing of the writes()of different process if we write in parallel. Parallel process will write in independent mode.

Read

Itterate over all images in sequence

This may well be the pattern used most often or almost exclusively. If analysis programs run in parallel each process or thread may read just a subset of images, but the subsets will be contiguous again. Even tomography can use this pattern with our new implementation of the sinogram generation. Some analysis software may combine several consecutive reads to a single read for performance reasons.

Strided access to images

For preview or a fast scans of the dataset we may access just every n^{th} image.

Access to regions of interest

I don't know whether we want to access regions of interests only, i.e. just a part of each image, this would happen again in a sequential order image after image.

Status today at SLS

High data volume and data rate beamlines – MX, cSAXS and Tomography

Five experimental stations (*beamlines*)

- three Macromolecular Crystallograpy (MX) beamlines *PX-I, PX-II, PX-III*
- the Micro-Tomography beamline *Tomcat* and

- the Coherent Small Angle X-Ray Scattering (*cSAXS*) beamline

create detector data at high data rates and produce large data volumes.

Current 2d detectors and cameras at SLS and other synchrotrons

We use Pilatus single photon counting detectors at MX and cSAXS beamlines and more conventional cameras at Tomcat:

- Pilatus6M-F: PX-I, PX-II
- Pilatus2M-F: PX-III
- Pilatus2M, Pilatus300K-W, Pilatus100K: cSAXS
- PCO.edge, PCO.2000, PCO.Dimax: Tomcat

Image size varies from 100K pixels for a Pilatus100K to 6M pixels for a Pilatus6M. All Pilatus detectors write 32bit unsigned data, PCO cameras write 16bit unsigned integer data. Uncompressed image size varies from 400KB to 24MB, predominantly 8MB to 24MB. File size (with possibly compressed image data) varies from 100KB to 10MB , mostly 2, 6, 8 or 10MB

Dedicated single server per detector at the beamline close the detector

Each detector is connected to a single dedicated detector server via a dedicated link (Cameralink for PCO cameras, GigaSTAR for Pilatus).
Detector servers have a 10GbE (or 1GbE) link for data transfer to the beamline.
Detector servers either write directly to the beamline storage or send image data via some other protocol (rsync, message passing) to some second server which writes to the beamline storage in turn.

One file per image

We store each detector image in a single separate file. Formats are either cbf for Pilatus or tiff for PCO.

Little to no metadata in image files

The image files in cbf or tiff format hold little to no metadata, all processing and analysis programs need input from other sources to get all required metadata.

4-fold compression for Pilatus data

Pilatus data in cbf format gets 4-times compressed on the detector server bys cbf's byte-offset compression.

Beamline data storage serves a central data hub

All applications that need to access the detector data read it from the beamline data storage. The data storage serves as central data hub even for those applications that run in parallel to the data acquisition:

- Preview of images at 1-10Hz
- Quality control – calculate and display some key values for each image (like center of mass, average and maximum intensity, number and quality of reflections, intensity histogram,…
- Initial processing or reconstruction
- Export data to removable disks on the beamline's media station or via network to the beamline user's home site.
- Pack (.tar, .tgz) and archive data to our tape archive