Smaller and Faster: Data Compression in areaDetector

Slides: Mark Rivers

GeoSoilEnviroCARS, Advanced Photon Source

University of Chicago

EPICS meeting, ITER, June 2019

Presenter: Ulrik Kofoed Pedersen, Head of Beamline Controls

Demo: Gary Yendell

Diamond Light Source

Outline

- ADCore releases since 2018: R3-4, R3-5, R3-6
- This talk:
  - New data compression features
  - Demo of data compression
  - Other major features from these releases

Data Compression Motivation

- We are already in the era of “big data” with existing detectors. Eiger, Pilatus, Lambda, PCO, FLIR/Point Grey, Xspress 3, etc.
  - Can all produce data faster than most disk systems can handle
  - All exceed 1 Gbit network capacity, and some exceed 10 Gbit.
  - Rapidly fill up disks
- Will become a more serious issue with synchrotron upgrades
  - Increased count rates will allow existing detectors to run at their maximum speed
  - New generations of even faster detectors will be coming
- Data compression can help with these issues
  - Must be fast and easy to use

Support for Compressed NDArrays and NTNDArrays

- NDArray has 2 new fields to support compression
- .codec field (struct Codec_t) to describe the compressor

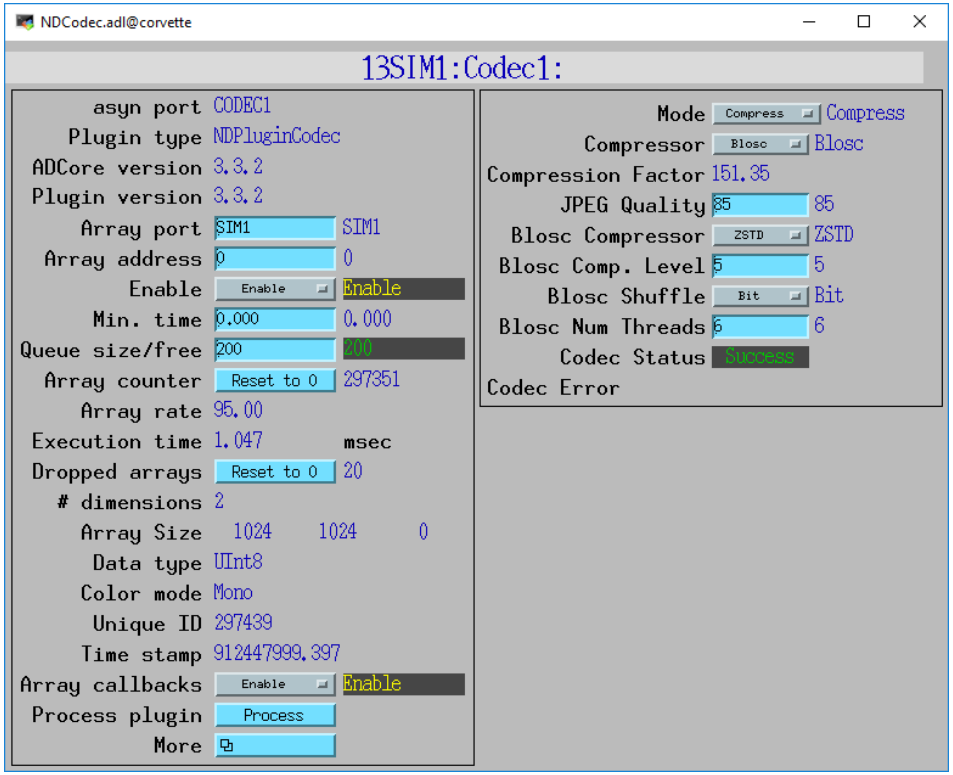
```
typedef enum {
  NDCODEC_NONE,
  NDCODEC_JPEG,
  NDCODEC_BLOSC,
  NDCODEC_LZ4,
  NDCODEC_BSLZ4
} NDCodecCompressor_t;

typedef struct Codec_t {
  std::string name;       /**< Name of the codec */
  int         level;      /**< Compression level. */
  int         shuffle;    /**< Shuffle type. */
  int         compressor; /**< Compressor type */
```

- .compressedSize (size_t) field with compressed size if codec.name is not empty.
- pvAccess NTNDArray has always had .compressedSize and .codec fields, but never previously implemented in servers or clients

NDPluginCodec (R3-4)

- New plugin for data compression and decompression
- Written by Bruno Martins from FRIB
- Mode:
  - Compress or Decompress
- Compressor:
  - None
  - JPEG (JPEGQuality selection)
  - Blosc (many options, next slide)
  - LZ4
  - BSLZ4 (Bitshuffle/lz4)
- CompFactor_RBV:
  - Actual compression ratio
- CodecStatus, CodecError
- JPEG is lossy, all others lossless



Blosc Codec Options

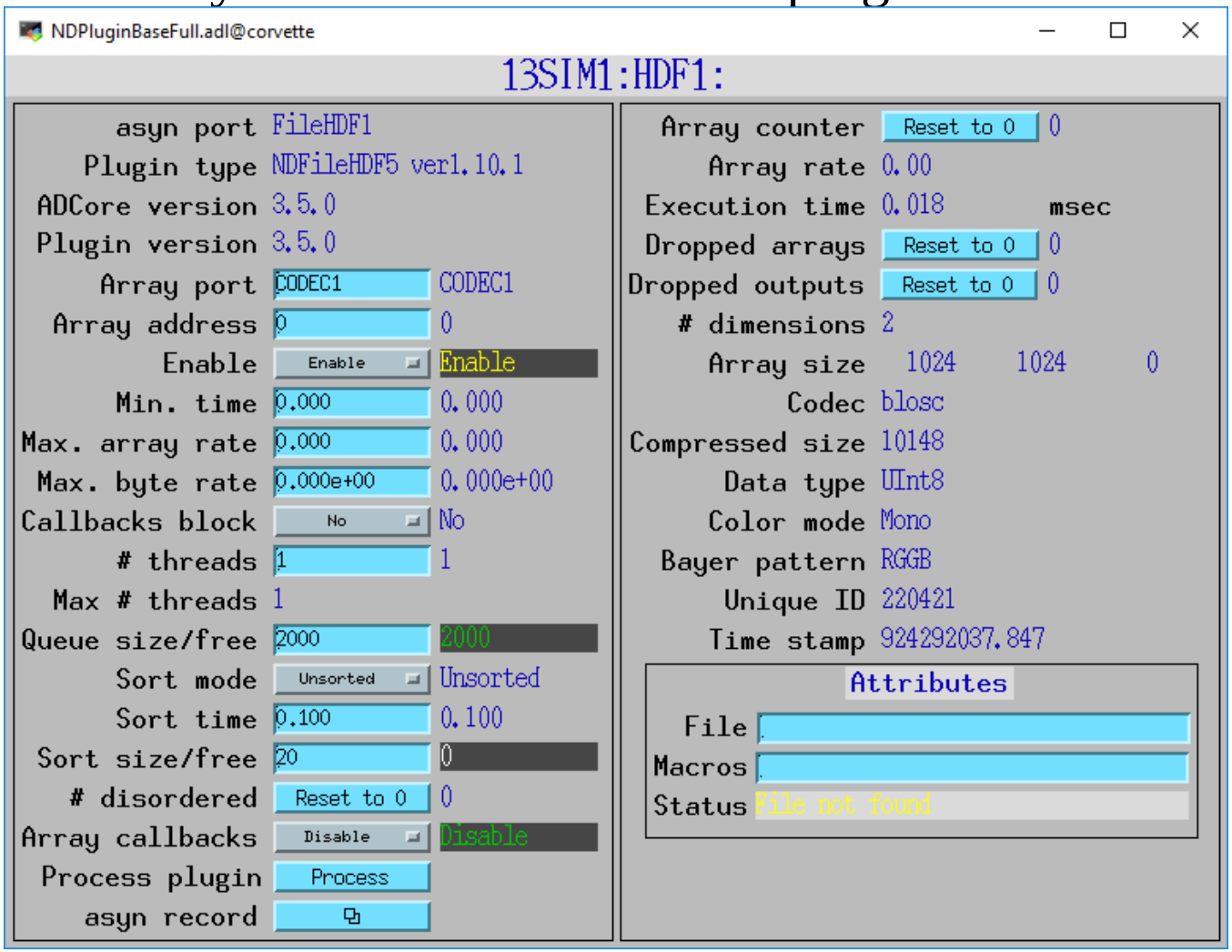
- BloscCompressor options. Each has different compression performance and speed
  - BloscLZ
  - LZ4
  - LZ4HC
  - Snappy
  - Zlib
  - Zstd
- BloscCLevel
  - Compression level: 0=no compression, 9=maximum compression.
  - Increasing execution time with increasing level.
- BloscShuffle
  - Choices = None, Byte, Bit.
  - Differences in speed and compression performance.
- BloscNumThreads
  - Number of threads used to compress each NDArray

LZ4 and BSLZ4 Codecs

- These are the codecs used by the Eiger server from Dectris
  - They don’t use the Blosc codecs, but rather the native LZ4 and Bitshuffle/LZ4 codecs.
- Dectris server can optionally use these compressions for HDF5 files saved locally on their server
- Dectris server always uses one of these compressions for data streamed over the ZeroMQ socket interface to the ADEiger driver
- These can now be decoded directly in ADEiger, or passed as compressed NDArrays to NDPluginCodec and other plugins
- Compressed arrays can be passed directly to NDFileHDF5 to be written with newly supported *direct chunk write* feature. More on this later.

Codec Parameter Records (R3-5)

- Codec_RBV and CompressedSize_RBV records to asynNDArrayDriver and hence to all plugins.



HDF5 Changes (R3-5 & R3-6)

- NDFileHDF5 file writing plugin has always supported the "built-in" compression filters from HDF5:
  - N-bit
  - SZIP
  - ZLIB
- HDF5 Dynamically Loadable Filters (ref)
  - R3-3 added support for Blosc filters
    - Thanks Xiaoqiang Wang, PSI
  - New support for LZ4 and Bitshuffle/LZ4 filters
  - All of these compressors are called from the HDF5 library.
    - Limits performance because of the overhead of the library.
- New support for HDF5 "Direct Chunk Write" *(ref)* **(Use R3-6)**
  - The NDArrays can be pre-compressed, either in NDPluginCodec, or directly by the driver (e.g. ADEiger)
  - Much faster, much of the code in the HDF5 library is skipped.
  - Thanks Gary Yendell, Diamond Light Source
- Fixed a number of memory leaks, some were significant
- Added FlushNow record to force flushing datasets to disk in SWMR mode

HDF5 Direct Chunk Write Performance

- 1024x1024 32-bit images
- simDetector generating ~1350 frames/s = 5.4 GB/s.
- Blosc LZ4 ByteShuffle compression
  - Compression level 6.
  - NDPluginCodec
  - 6 Blosc threads
  - 3 plugin threads.
  - Compression factor is ~64
- Time to save a single HDF5 file with 10,000 frames.

	No compression	NDFileHDF5 compression	NDPluginCodec compression, direct chunk write
File size (MB)	40,000	650	650
Total time (s)	106	32	7.4
Frame/s	94	312	1,351
MB/s uncompressed	389	1,250	5,405
MB/s compressed	N.A.	20	88

- HDF5 library can only compress 312 frames/s
- NDPluginCodec & direct chunk write keeps up with simDetector 1,350 frames/s

HDF5 Decompression Plugin Filters (ADSupport R1-7)

- HDF5 supports dynamic loading of compression and decompression filter libraries at run time.
- The Blosc, LZ4 and BSLZ4 have been built into the HDF5 library in ADSupport so that dynamic loading is *not* required when using NDFileHDF5.
- However, to decompress HDF5 files compressed with Blosc, LZ4 or BSLZ4 with other applications dynamic loading of the filters will be required
- ADSupport now builds these dynamic filter libraries for Linux, Windows, and Mac.
- Must set the following environment variable to use them:

```
HDF5_PLUGIN_PATH=[areaDetector]/ADSupport/lib/linux-x86_64
```

HDF5 Decompression Plugin Filters

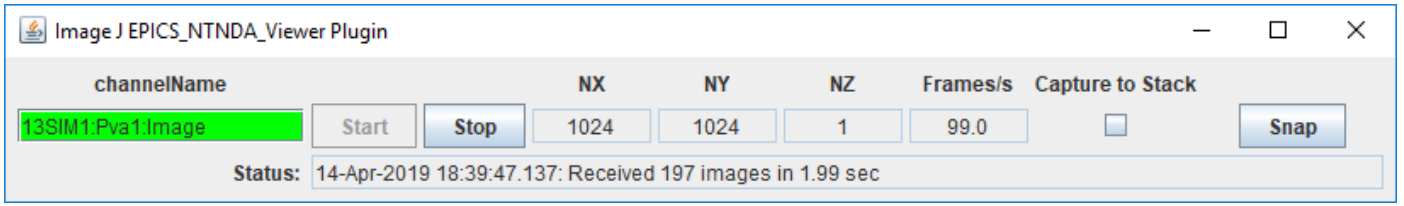
```
>h5dump --properties test_hdf5_direct_chunk_3.h5
HDF5 "test_hdf5_direct_chunk_3.h5" {
GROUP "/" {
   GROUP "entry" {
…
         DATASET "data" {
            DATATYPE  H5T_STD_U32LE
            DATASPACE  SIMPLE { ( 100, 1024, 1024 ) / ( 100, 1024, 1024 ) }
            STORAGE_LAYOUT {
               CHUNKED ( 1, 1024, 1024 )
               SIZE 4082368 (102.742:1 COMPRESSION)
            }
            FILTERS {
               USER_DEFINED_FILTER {
                  FILTER_ID 32001
                  COMMENT blosc
                  PARAMS { 2 2 4 4194304 8 1 1 }
               }
            }
            …
            DATA {
            (0,0,0): 173140, 173141, 173142, 173143, 173144, 173145, 173146,
            (0,0,7): 173147, 173148, 173149, 173150, 173151, 173152, 173153,
            (0,0,14): 173154, 173155, 173156, 173157, 173158, 173159, 173160,
            (0,0,21): 173161, 173162, 173163, 173164, 173165, 173166, 173167,
```

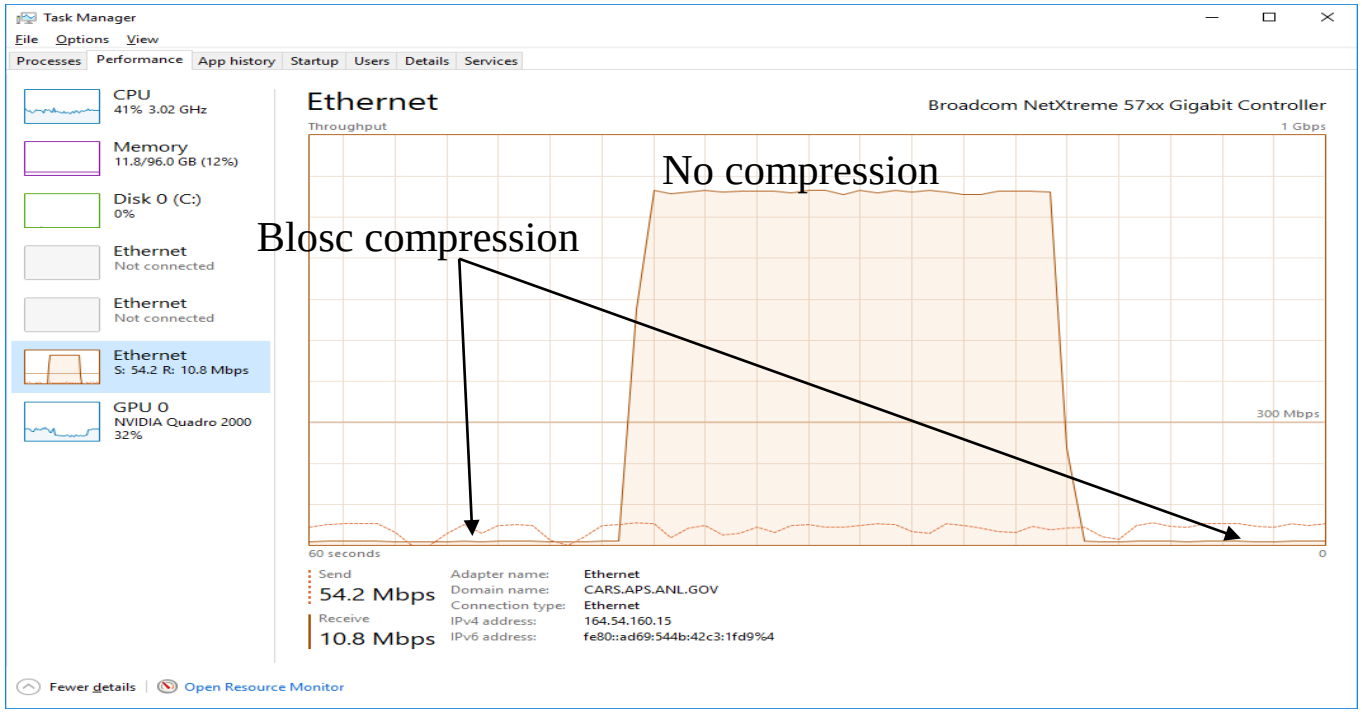
Demo

Compression

ImageJ pvAccess Viewer

- Now supports displaying compressed NTNDArrays
- Supports all compressions (JPEG, Blosc, LZ4, BSLZ4)
- Can greatly reduce network bandwidth when the IOC and viewer are running on different machines

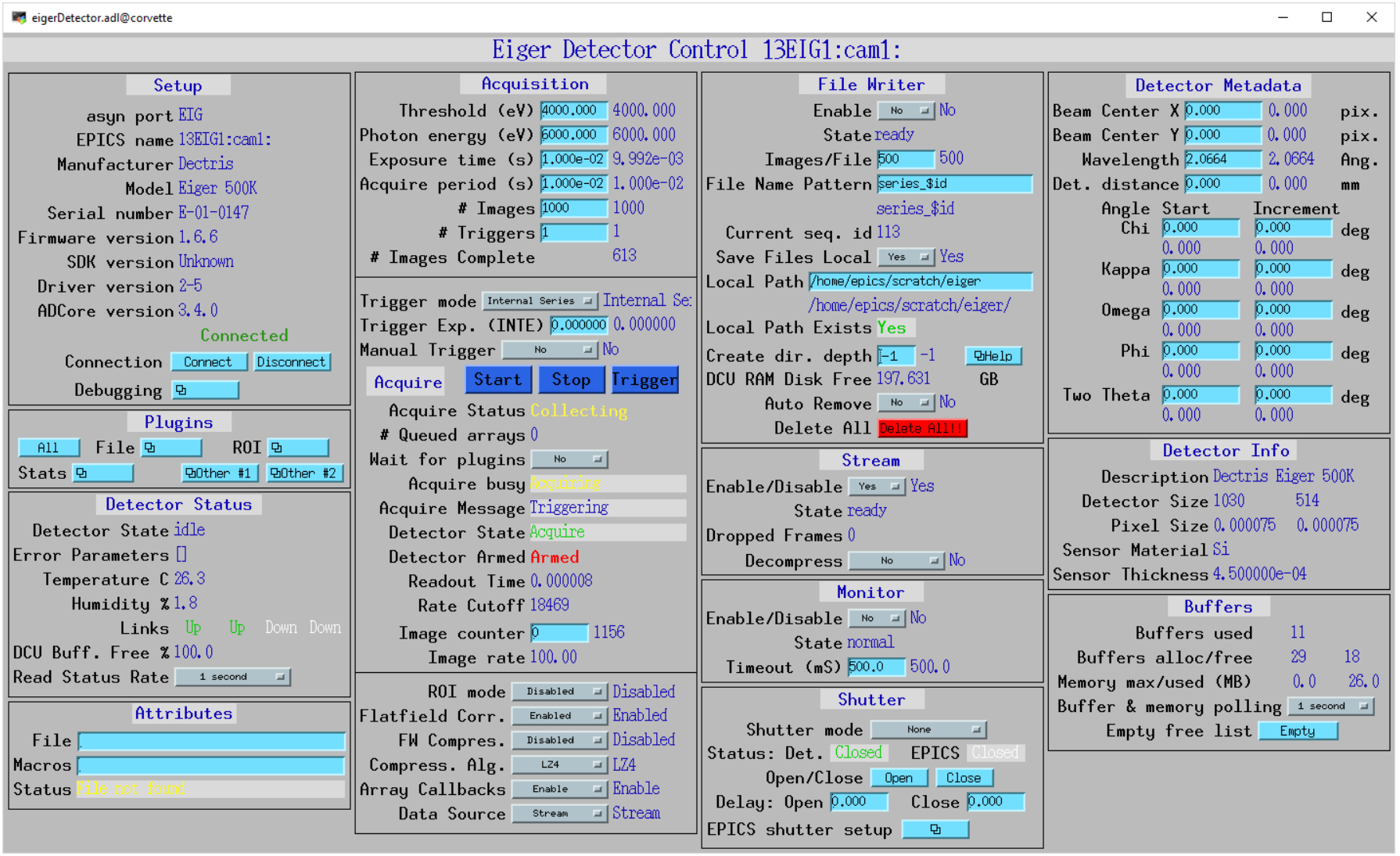




ADEiger Changes

- Now supports Bitshuffle/LZ4 on Stream interface over ZeroMQ
  - Previously only LZ4 was supported
- New StreamDecompress bo record to enable/disable decompression. If disabled:
  - NDFileHDF5 can use Direct Chunk Write without ever decompressing
  - NDPluginPva can send to ImageJ without ever decompressing
  - NDPluginCodec can decompress for other plugins like NDPluginStats, etc.
- Eiger Simplon API version 1.6 -> 1.8 changes WIP
  - https://github.com/areaDetector/ADEiger/pull/27

ADEiger Changes



Improvements in ADCore (R3-4)

- New MaxByteRate record for plugins to limit “output rate”
  - For most plugins this limits the byte rate of the NDArrays passed to downstream plugins
  - For NDPluginStdArrays it limits the byte rate of the data callbacks to waveform records, and hence to Channel Access clients
  - For NDPluginPva it limits the byte rate of the data callbacks to pvAccess clients
- Optimization improvement when output arrays are sorted.
  - Previously it always put the array in the sort queue, even if the order of this array was OK.
  - Introduced an unneeded latency because the sort task only runs periodically.
  - Caused ImageJ update rates to be slow, because it made PVA output comes in bursts, and some arrays were dropped either in the pvAccess server or client.
  - Now if the array is in the correct order it is output immediately.

Documentation Improvements (R3-5)

- Documentation was changed from manually edited HTML pages to reStructuredText (.rst) files processed with Spinx.
  - Most tables are left in native HTML because .rst conversion is poor quality
- Server changed from https://cars.uchicago.edu/software/epics/ to areaDetector.github.io/
- Advantages:
  - Easier to edit
  - Nicer looking pages
- New Travis CI job at top-level areaDetector
  - Runs doxygen and sphinx to update the areaDetector.github.io files every time there is a push to the top-level areaDetector repository.
- Thanks to Stuart Wilkins from BNL who set up the process and converted all of the files in ADCore, ADProsilica, and ADFastCCD.
- Other detector repositories still need to be converted.
  - Use pandoc to convert .html to .rst. Manual editing still required.

areaDetector Plugin NDPluginProcess

April 16, 2018

Mark Rivers

University of Chicago

Contents

Overview

NDPluginProcess performs arithmetic processing on NDArray data. It performs the following operations in the order listed. Each of these operations can be individually enabled and disabled.

1. Subtracts a background array which has been previously acquired.
2. Divides by a flat field array which has been previously acquired, and then multiplies by a flat field scale factor.
3. Multiplies by a scale factor and adds an offset.
4. Clips to a maximum specified value.
5. Clips to a minimum specified value.
6. Applies a recursive digital filter.
7. Converts to the specified output data type.
8. Exports the processed data as a new NDArray object.

If any of the above operations is enabled, then the array is first converted to NDFloat64 data type, i.e. double-precision float. The operations are all performed in double-precision, and then the array is converted to the specified output data type.

NDPluginProcess is both a **recipient** of callbacks and a **source** of NDArray callbacks. This means that other plugins, such the NDPluginStdArrays, NDPluginStats, and NDPluginFile plugins can be connected to an NDPluginProcess plugin, in which case they will use the processed data.

NDPluginProcess is fully N-dimensional. It can be used for 2-D images, 3-D (color) images, or any type of N-dimensional data.

NDPluginProcess inherits from NDPluginDriver. The NDPluginProcess class documentation describes this class in detail.

NDPluginProcess.h defines the following parameters. It also implements all of the standard plugin parameters from NDPluginDriver. The EPICS database NDProcess.template provide access to these parameters, listed in the following table. Note that to reduce the width of this table the parameter index variable names have been split into 2 lines, but these are just a single name, for example `NDPluginProcessSaveBackground`.

Parameter Definitions in NDPluginProcess.h and EPICS Record Definitions in NDProcess.template						
Parameter index variable	**asyn interface**	**Access**	**Description**	**drvInfo string**	**EPICS record name**	**EPICS record type**
Background subtraction						
NDPluginProcess SaveBackground	asynInt32	r/w	Command to use the most recently acquired array as a background. Note that this recently acquired array should have been acquired with EnableBackground=0, or else that array will already have had the background subtracted, which is probably not what was intended!	SAVE_BACKGROUND	(P)(R)SaveBackground (P)(R)SaveBackground_RBV	bo bi

NDPluginProcess

author: Mark Rivers, University of Chicago

Contents

Overview

NDPluginProcess performs arithmetic processing on NDArray data. It performs the following operations in the order listed. Each of these operations can be individually enabled and disabled.

- Subtracts a background array which has been previously acquired.
- Divides by a flat field array which has been previously acquired, and
- then multiplies by a flat field scale factor.
- Multiplies by a scale factor and adds an offset.
- Clips to a maximum specified value.
- Clips to a minimum specified value.
- Applies a recursive digital filter.
- Converts to the specified output data type.
- Exports the processed data as a new NDArray object.

If any of the above operations is enabled, then the array is first converted to NDFloat64 data type, i.e. double-precision float. The operations are all performed in double-precision, and then the array is converted to the specified output data type.

NDPluginProcess is both a **recipient** of callbacks and a **source** of NDArray callbacks. This means that other plugins, such the NDPluginStdArrays, NDPluginStats, and NDPluginFile plugins can be connected to an NDPluginProcess plugin, in which case they will use the processed data.

NDPluginProcess is fully N-dimensional. It can be used for 2-D images, 3-D (color) images, or any type of N-dimensional data.

NDPluginProcess inherits from NDPluginDriver. The NDPluginProcess class documentation describes this class in detail.

NDPluginProcess.h defines the following parameters. It also implements all of the standard plugin parameters from NDPluginDriver. The EPICS database NDProcess.template provide access to these parameters, listed in the following table. Note that to reduce the width of this table the parameter index variable names have been split into 2 lines, but these are just a single name, for example `NDPluginProcessSaveBackground`.

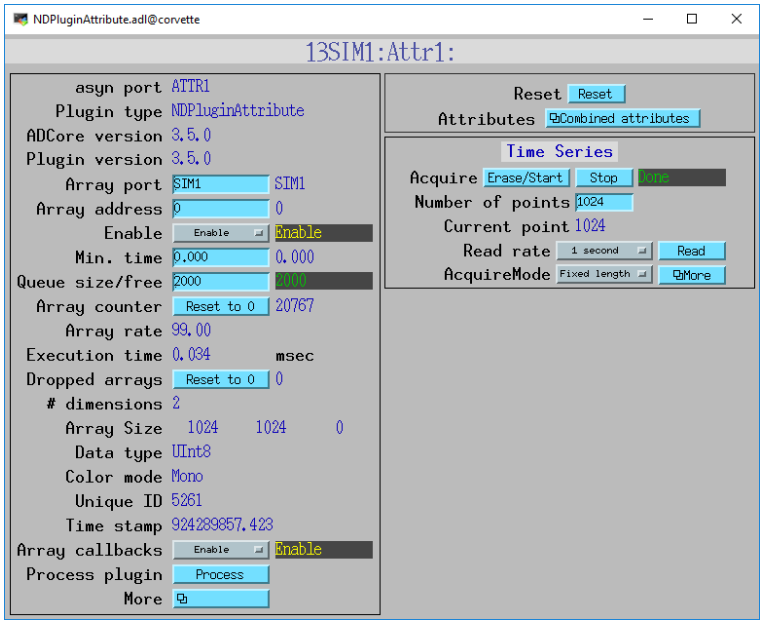
Parameter Definitions in NDPluginProcess.h and EPICS Record Definitions in NDProcess.template

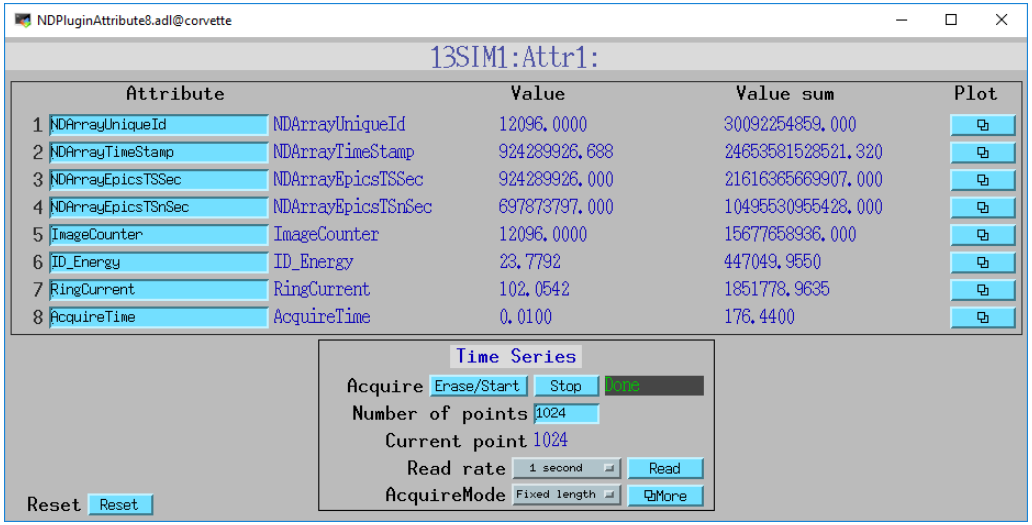
Parameter index variable	asyn interface	Access	Description	drvInfo string	EPICS record name	EPICS record type
Background subtraction						
NDPluginProcess SaveBackground	asynInt32	r/w	Command to use the most recently acquired array as a background. Note that this recently acquired array should have been acquired with EnableBackground=0, or else that array will already have had the background subtracted, which is	SAVE_BACKGROUND	(P)(R)SaveBackground (P)(R)SaveBackground_RBV	bo bi

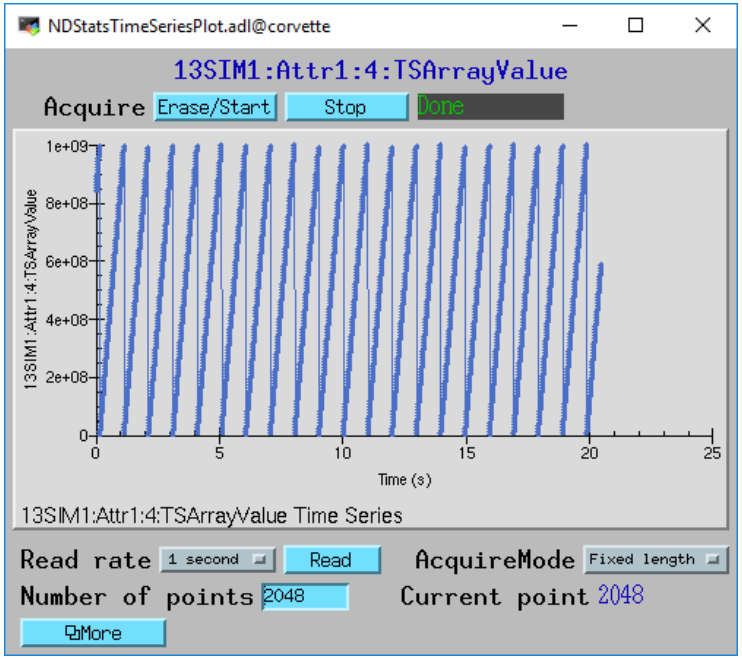
NDPluginAttribute Time Series (R3-5)

- Previously NDPluginAttribute time series code was internal
- Changed so that it now uses NDPluginTimeSeries, same change that was made to NDPluginStats in R3-5
- Fewer lines of code, and adds Circular Buffer mode
- The time-series waveform PVs are the same
- The PVs to control the time-series (start/stop, # of points) have changed, so clients may need modifications

NDPluginAttribute Time Series (R3-5)







Roadmap: ADCore R5-0?

- Use NTNDArrays inside drivers and plugins
- Use pvDatabase
  - “local” provider within IOC
  - “pva” provider between IOCs
- Smart pointers automatically eliminate all unnecessary copying
- Eliminates need for NDPluginPva
- V4 clients can immediately receive data from any point in plugin chain
- Distribute load to multiple IOCs without pvaDriver
- Bruno Martins has demonstrated this working for ADSimDetector and NDPluginStats