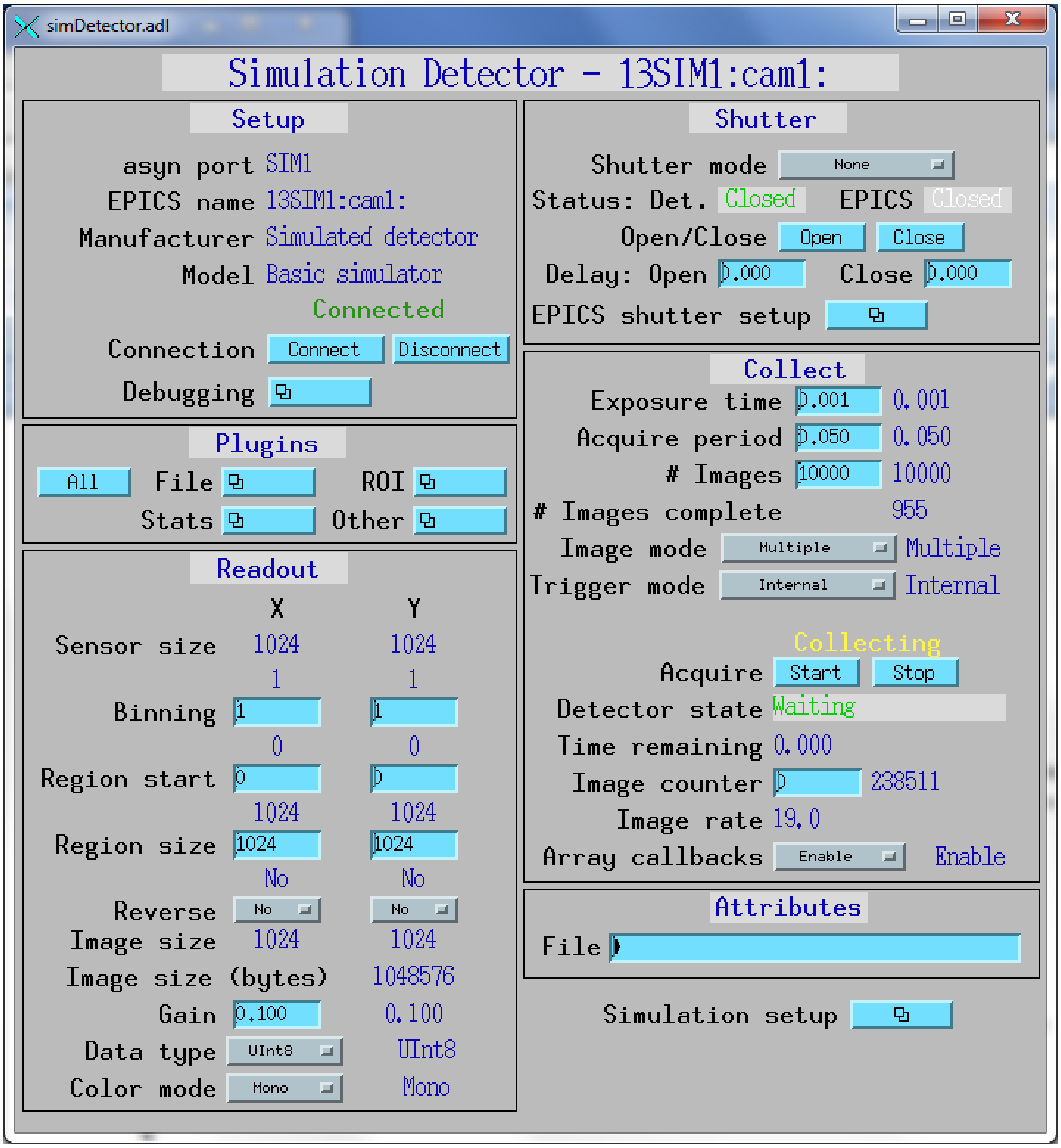
Measurement of Performance of areaDetector Plugin Architecture

Mark Rivers

June 21, 2013

These measurements were designed to determine how effectively the areaDetector plugins use multiple cores on multi-core Linux and Windows systems. The measurements were made using the SVN revision 16620 or areaDetector, which was the HEAD version on this date. This is essentially the same as areaDetector R1-9-1.

The simDetector was used to produce images at varying rates. The simDetector was configured to produce 1024x1024 8-bit monochrome images. The AcquireTime was 0.001 s, and the AcquirePeriod was adjusted to control the frame rate. The ImageMode was “Multiple”, and a fixed number of frames were acquired for each test. No XML attribute file was used. The configuration of the simDetector is shown in the following medm screen shot:



simDetector.adl

Simulation Detector - 13SIM1:cam1:

Setup

asyn port	SIM1
EPICS name	13SIM1:cam1:
Manufacturer	Simulated detector
Model	Basic simulator
	Connected
Connection	Connect / Disconnect
Debugging	

Plugins

All · File · ROI · Stats · Other

Readout

	X	Y
Sensor size	1024	1024
Binning	1 / 1	1 / 1
Region start	0 / 0	0 / 0
Region size	1024 / 1024	1024 / 1024
Reverse	No / No	No / No
Image size	1024	1024
Image size (bytes)		1048576
Gain	0.100	0.100
Data type	UInt8	UInt8
Color mode	Mono	Mono

Shutter

Shutter mode	None
Status: Det.	Closed
EPICS	Closed
Open/Close	Open / Close
Delay: Open	0.000
Close	0.000
EPICS shutter setup	

Collect

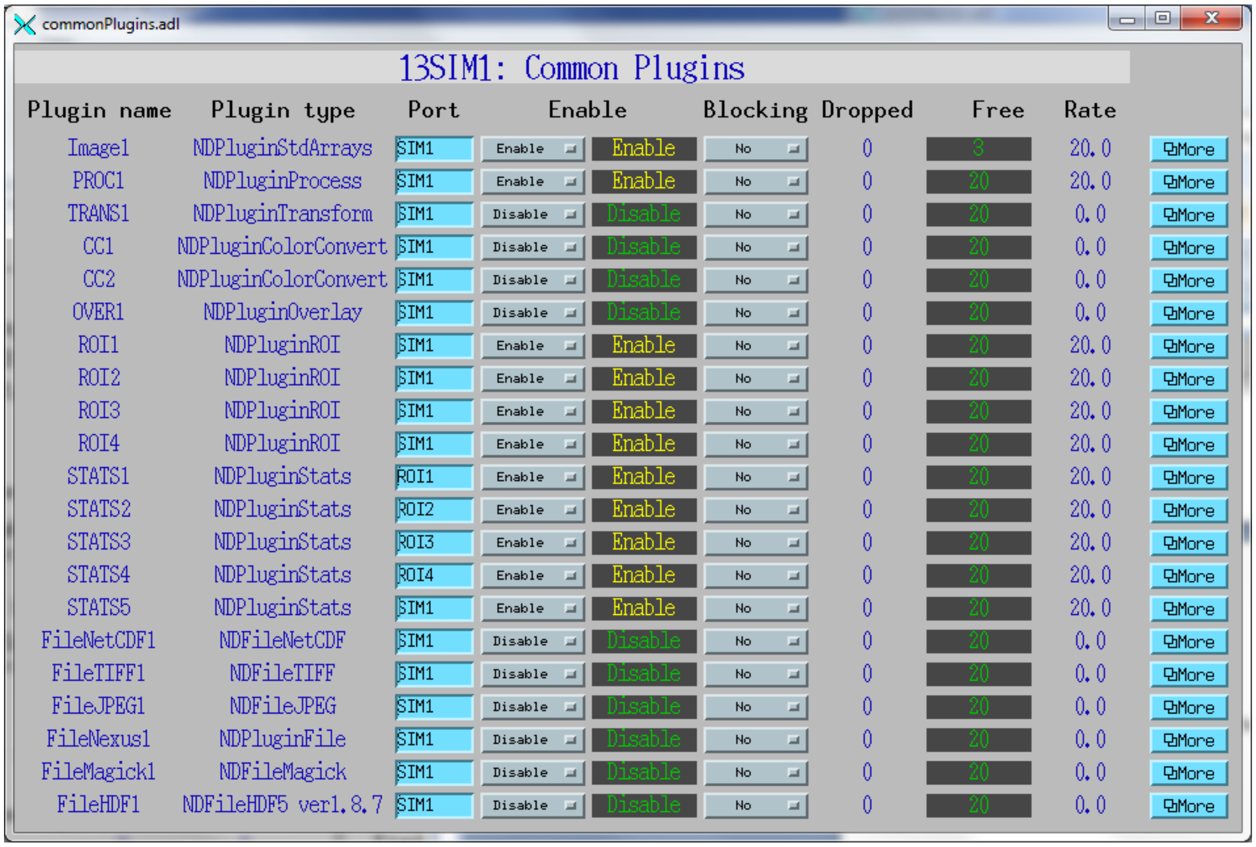
Exposure time	0.001	0.001
Acquire period	0.050	0.050
# Images	10000	10000
# Images complete		955
Image mode	Multiple	Multiple
Trigger mode	Internal	Internal
		Collecting
Acquire	Start / Stop	
Detector state	Waiting	
Time remaining	0.000	
Image counter	0	238511
Image rate	19.0	
Array callbacks	Enable	Enable

Attributes

File

Simulation setup

For the first set of measurements the following plugin configuration was used.



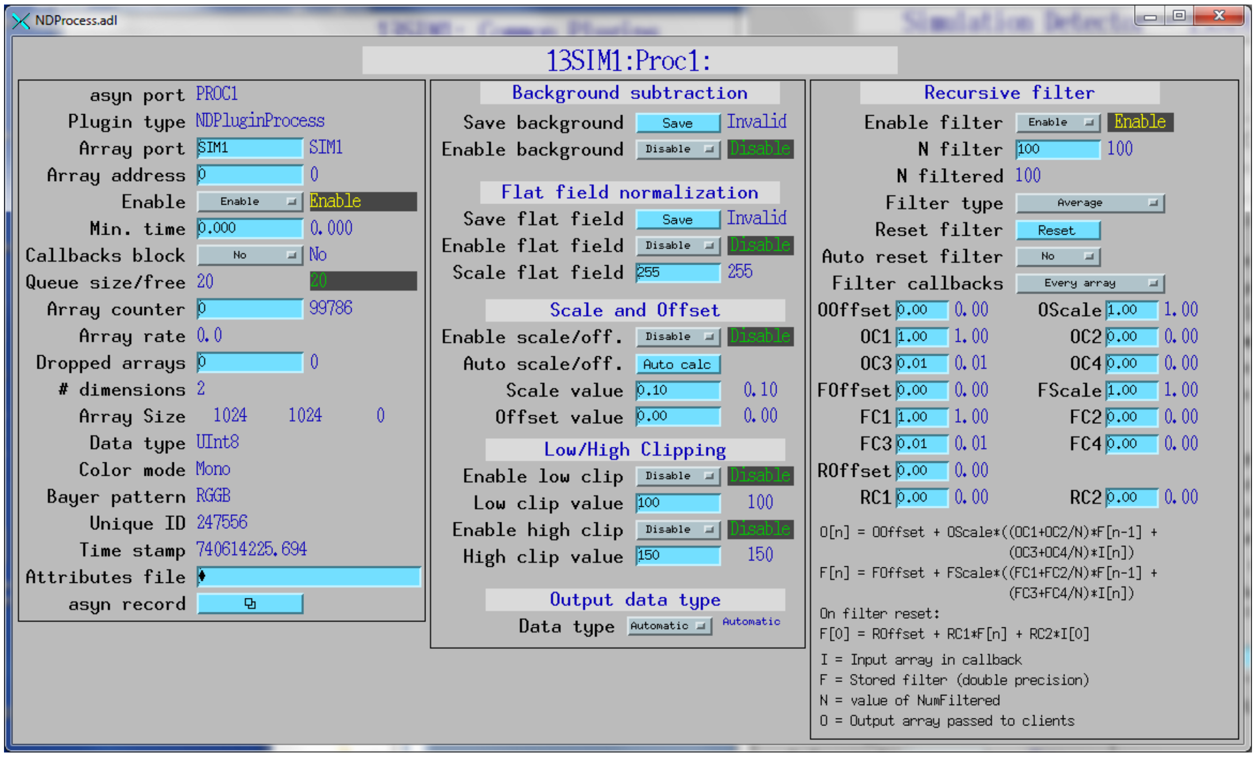
commonPlugins.adl

13SIM1: Common Plugins

Plugin name	Plugin type	Port	Enable		Blocking	Dropped	Free	Rate	
Image1	NDPluginStdArrays	SIM1	Enable	Enable	No	0	3	20.0	More
PROC1	NDPluginProcess	SIM1	Enable	Enable	No	0	20	20.0	More
TRANS1	NDPluginTransform	SIM1	Disable	Disable	No	0	20	0.0	More
CC1	NDPluginColorConvert	SIM1	Disable	Disable	No	0	20	0.0	More
CC2	NDPluginColorConvert	SIM1	Disable	Disable	No	0	20	0.0	More
OVER1	NDPluginOverlay	SIM1	Disable	Disable	No	0	20	0.0	More
ROI1	NDPluginROI	SIM1	Enable	Enable	No	0	20	20.0	More
ROI2	NDPluginROI	SIM1	Enable	Enable	No	0	20	20.0	More
ROI3	NDPluginROI	SIM1	Enable	Enable	No	0	20	20.0	More
ROI4	NDPluginROI	SIM1	Enable	Enable	No	0	20	20.0	More
STATS1	NDPluginStats	ROI1	Enable	Enable	No	0	20	20.0	More
STATS2	NDPluginStats	ROI2	Enable	Enable	No	0	20	20.0	More
STATS3	NDPluginStats	ROI3	Enable	Enable	No	0	20	20.0	More
STATS4	NDPluginStats	ROI4	Enable	Enable	No	0	20	20.0	More
STATS5	NDPluginStats	SIM1	Enable	Enable	No	0	20	20.0	More
FileNetCDF1	NDFileNetCDF	SIM1	Disable	Disable	No	0	20	0.0	More
FileTIFF1	NDFileTIFF	SIM1	Disable	Disable	No	0	20	0.0	More
FileJPEG1	NDFileJPEG	SIM1	Disable	Disable	No	0	20	0.0	More
FileNexus1	NDPluginFile	SIM1	Disable	Disable	No	0	20	0.0	More
FileMagick1	NDFileMagick	SIM1	Disable	Disable	No	0	20	0.0	More
FileHDF1	NDFileHDF5 ver1.8.7	SIM1	Disable	Disable	No	0	20	0.0	More

The Image1, PROC1, ROI[1-4], STATS[1-5] plugins were enabled, and all other plugs were disabled. All file writing plugins were disabled.

The PROC1 plugin was configured to get its data from the entire detector (SIM1) and to average 100 frames as shown below.



NDProcess.adl

13SIM1:Proc1:

asyn port	PROC1	
Plugin type	NDPluginProcess	
Array port	SIM1	SIM1
Array address	0	0
Enable	Enable	Enable
Min. time	0.000	0.000
Callbacks block	No	No
Queue size/free	20	20
Array counter	0	99786
Array rate	0.0	
Dropped arrays	0	0
# dimensions	2	
Array Size	1024 1024 0	
Data type	UInt8	
Color mode	Mono	
Bayer pattern	RGGB	
Unique ID	247556	
Time stamp	740614225.694	
Attributes file		
asyn record		

Background subtraction

Save background	Save	Invalid
Enable background	Disable	Disable

Flat field normalization

Save flat field	Save	Invalid
Enable flat field	Disable	Disable
Scale flat field	255	255

Scale and Offset

Enable scale/off.	Disable	Disable
Auto scale/off.	Auto calc	
Scale value	0.10	0.10
Offset value	0.00	0.00

Low/High Clipping

Enable low clip	Disable	Disable
Low clip value	100	100
Enable high clip	Disable	Disable
High clip value	150	150

Output data type

Data type	Automatic	Automatic

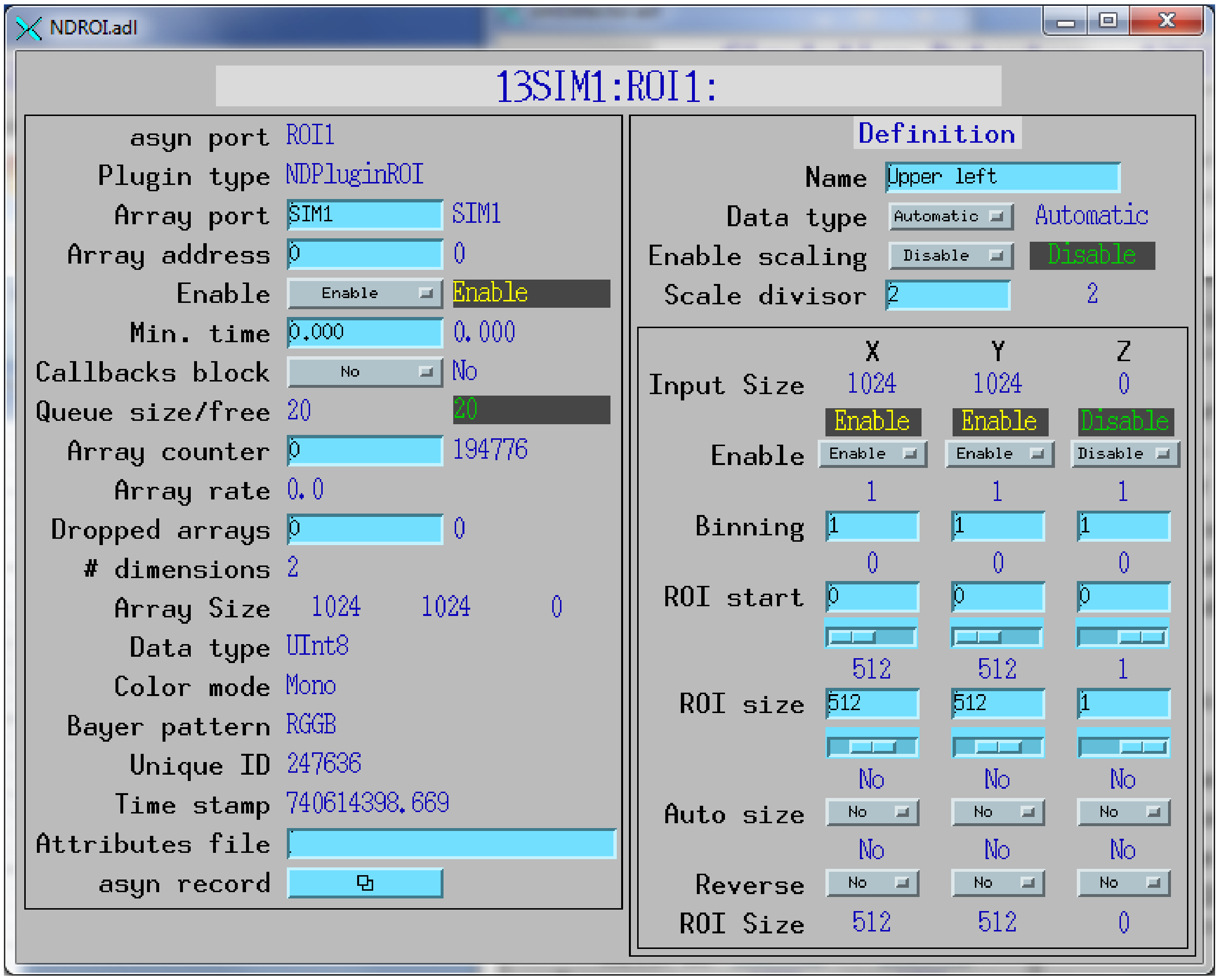
Recursive filter

Enable filter	Enable	Enable
N filter	100	100
N filtered	100	
Filter type	Average	
Reset filter	Reset	
Auto reset filter	No	
Filter callbacks	Every array	

OOffset	0.00	0.00	OScale	1.00	1.00
OC1	1.00	1.00	OC2	0.00	0.00
OC3	0.01	0.01	OC4	0.00	0.00
FOffset	0.00	0.00	FScale	1.00	1.00
FC1	1.00	1.00	FC2	0.00	0.00
FC3	0.01	0.01	FC4	0.00	0.00
ROffset	0.00	0.00			
RC1	0.00	0.00	RC2	0.00	0.00

```
O[n] = OOffset + OScale*((OC1+OC2/N)*F[n-1] +
                         (OC3+OC4/N)*I[n])
F[n] = FOffset + FScale*((FC1+FC2/N)*F[n-1] +
                         (FC3+FC4/N)*I[n])
On filter reset:
F[0] = ROffset + RC1*F[n] + RC2*I[0]
I = Input array in callback
F = Stored filter (double precision)
N = value of NumFiltered
O = Output array passed to clients
```

The ROI plugins were each configured to extract a 512x512 subset of the images, each extracting one of the 4 quadrants of the simDetector. Scaling, Auto-Size, and Reverse were disabled. For example ROI1 was configured as follows:



NDROI.adl

13SIM1:ROI1:

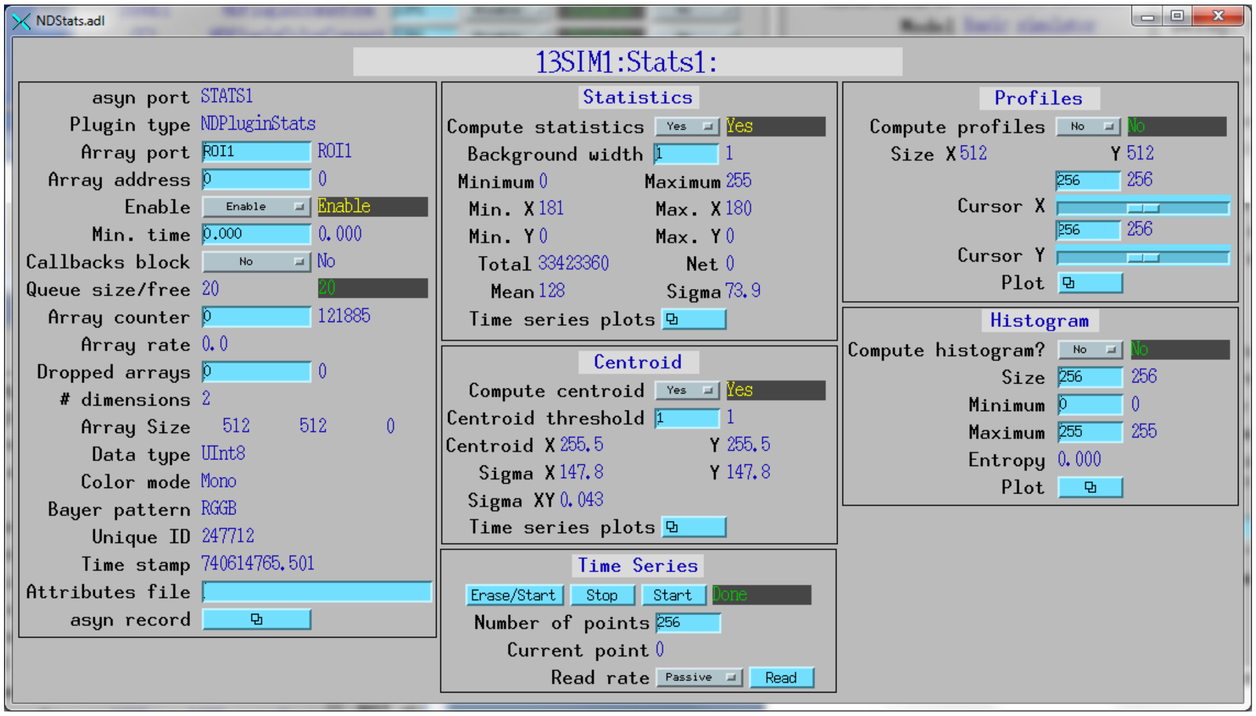
asyn port	ROI1	
Plugin type	NDPluginROI	
Array port	SIM1	SIM1
Array address	0	0
Enable	Enable	Enable
Min. time	0.000	0.000
Callbacks block	No	No
Queue size/free	20	20
Array counter	0	194776
Array rate	0.0	
Dropped arrays	0	0
# dimensions	2	
Array Size	1024 1024 0	
Data type	UInt8	
Color mode	Mono	
Bayer pattern	RGGB	
Unique ID	247636	
Time stamp	740614398.669	
Attributes file		
asyn record		

Definition

Name	Upper left	
Data type	Automatic	Automatic
Enable scaling	Disable	Disable
Scale divisor	2	2

	X	Y	Z
Input Size	1024	1024	0
Enable	Enable / Enable	Enable / Enable	Disable / Disable
Binning	1 / 1	1 / 1	1 / 1
ROI start	0 / 0	0 / 0	0 / 0
ROI size	512 / 512	512 / 512	1 / 1
Auto size	No / No	No / No	No / No
Reverse	No / No	No / No	No / No
ROI Size	512	512	0

The statistics plugins were configured to compute the basic statistics and the centroid, but not the profiles or histogram. STATS[1-4] get their data from ROI[1-4]. STATS5 gets its data from SIM1, the entire detector. The configuration of STATS1 is shown below.



13SIM1:Stats1:

asyn port	STATS1	
Plugin type	NDPluginStats	
Array port	ROI1	ROI1
Array address	0	0
Enable	Enable	Enable
Min. time	0.000	0.000
Callbacks block	No	No
Queue size/free	20	20
Array counter	0	121885
Array rate	0.0	
Dropped arrays	0	0
# dimensions	2	
Array Size	512 512 0	
Data type	UInt8	
Color mode	Mono	
Bayer pattern	RGGB	
Unique ID	247712	
Time stamp	740614765.501	
Attributes file		
asyn record		

Statistics

Compute statistics Yes Yes
Background width 1 1
Minimum 0 Maximum 255
Min. X 181 Max. X 180
Min. Y 0 Max. Y 0
Total 33423360 Net 0
Mean 128 Sigma 73.9
Time series plots

Centroid

Compute centroid Yes Yes
Centroid threshold 1 1
Centroid X 255.5 Y 255.5
Sigma X 147.8 Y 147.8
Sigma XY 0.043
Time series plots

Time Series

Erase/Start Stop Start Done
Number of points 256
Current point 0
Read rate Passive Read

Profiles

Compute profiles No No
Size X 512 Y 512
256 256
Cursor X
256 256
Cursor Y
Plot

Histogram

Compute histogram? No No
Size 256 256
Minimum 0 0
Maximum 255 255
Entropy 0.000
Plot

Performance Using Linux

Using the configuration shown above the performance was measured as a function of the AcquirePeriod, which controls the frame rate of the simDetector.

Table 1 summarizes the measurements on a Linux system with dual quad-core CPUS (Intel(R) Xeon(R) CPU, E5630 @ 2.53GHz). The system thus has 8 physical cores, but has hyper-threading enabled, and so has 16 virtual cores.

Table 1. Linux system with dual quad-core CPUS (Intel(R) Xeon(R) CPU, E5630 @ 2.53GHz)

Acquire Period	0.05	0.02	0.01	0.005	0.002
# Frames	1000	10000	10000	10000	10000
Detector Frames/s	20	50	99	196	440
CPU %	300	500	600	800	1050
Frames dropped					
Image1	0	0	0	0	241
PROC1	0	3790	6965	8617	9484
ROI1	0	0	0	0	7
ROI2	0	0	0	0	11
ROI3	0	0	0	0	0
ROI4	0	0	0	0	0
STATS1	0	0	0	3174	7555
STATS2	0	0	0	2961	7470
STATS3	0	0	0	2964	7453
STATS4	0	0	0	2938	7508
STATS5	146	6642	7547	8848	9582

At 20 frames/s only the STATS5 plugin drops any frames (about 15%), and the system is using 300% of the CPU, or 3 cores.

At 50 frames/s the PROC1 plugin is dropping about 28% of the frames and the STATS5 plugins is dropping about 66% of the frames. The system is using 500% of the CPU, or 5 cores.

At 99 frames/s the PROC1 plugin is dropping about 70% of the frames and the STATS5 plugins is dropping about 75% of the frames. The system is using 600% of the CPU, or 6 cores.

At 196 frames/s both the PROC1 and all of the STATS plugins are dropping 30%-88% of the frames, and the system is using 900% of the CPU, or 9 cores.

At 440 frames/s the Image1 and ROI plugins are also beginning to drop frames, and the system is using 1050% of the CPU, or 10.5 cores.

The above table shows close to ideal behavior as the frame rate increases. The rate at which the simDetector is generating frames is very close to the inverse of the AcquirePeriod. Even at 0.002 seconds when it should have generated 500 frames/s it is generating 440 frames/sec, which is 88% of the theoretical rate. This shows that the simDetector thread is not being slowed down by the plugins, which are saturating their threads and dropping up 75%-95% of the frames.

A simple view of the system is that there are 11 plugins, each running in its own thread, plus the simDetector thread that computes the images. When the system is running at its maximum possible rate there should thus be 12 cores running at 100% CPU, or 1200% CPU time in “top”. In fact we reached 1050% CPU, or 10.5 cores, and at the point the ROI threads were not saturated since they are dropping only a few frames.

Conclusions: the areaDetector plugin architecture and the Linux scheduler are not getting in the way of nearly ideal scaling as the frame rate increases.

Performance Using Windows

The Windows minimum thread sleep time is 0.01 second, so it is not possible to achieve frame rates above 65 frames/s until the sleep time is actually 0, at which time the system goes to 100% CPU utilization. Because of this I changed the configuration above slightly. The simDetector images were increased to 2048x2048, and the ROIs extracted 1024x1024 quadrants from the detector.

Table 2 summarizes the measurements on a Windows 7 64-bit computer system with dual quad-core CPUS (Intel(R) Core(TM) i7-2820QM CPU@ 2.30GHz). The system thus has 8 physical cores, and does not have hyper-threading, so has 8 cores total..

Table 2. Windows 7 64-bit computer system with dual quad-core CPUS (Intel(R) Core(TM) i7-2820QM CPU@ 2.30GHz).

Acquire Period	0.05	0.02	0.015	.005	0.001 (*AcquireTime=0)*
# Frames	2000	2000	10000	10000	10000
Detector Frames/s	19	38	55	65	150
CPU %	30	43	61	77	99
Frames dropped					
Image1	0	0	0		5951
PROC1	97	1068	7152	7781	9274
ROI1	0	0	0		371
ROI2	0	0	0		2283
ROI3	0	0	0		603
ROI4	0	0	0		2336
STATS1	0	0	0		5577
STATS2	0	0	0		2598
STATS3	0	0	0		5604
STATS4	0	0	0		2968
STATS5	0	786	6322	7134	9241

The CPU numbers reported on Windows is the percentage of all the cores, so 100% means all cores saturated. This would be equivalent to 800% on an 8 core Linux system

At 20 frames/s only the PROC1 plugin drops any frames (about 5%), and the system is using 30% of the CPU, or 2.4 cores.

At 38 frames/s the PROC1 plugin drops about 53% of the frames, and the STATS5 plugin drops about 39% of the frames. The system is using 43% of the CPU, or 3.4 cores.

At 55 frames/s the PROC1 plugin drops about 72% of the frames, and the STATS5 plugin drops about 63% of the frames. The system is using 61% of the CPU, or 4.9 cores.

At 65 frames/s the PROC1 plugin drops about 78% of the frames, and the STATS5 plugin drops about 71% of the frames. The system is using 77% of the CPU, or 6.2 cores.

In order to achieve a very high frame rate I had to set AcquireTime=0 and AcquirePeriod=.001. Under these conditions the system is 100% CPU busy. But it can still be seen that the simDetector thread is not being held up by the plugins. As on Linux the PROC1 and STATS5 plugins are dropping over 90% of the frames, but other plugins and the simDetector main thread are not being held back by these plugins.

Conclusion: the areaDetector plugin architecture and the Windows scheduler are not getting in the way of nearly ideal scaling as the frame rate increases.

Performance with File Plugins Enabled

These tests were all done under the conditions detailed for Windows above, i.e. 2048x2048 frames, 8-bit. The ROI plugins were each, extracting a 1024x1024 subregion., each a different quadrant.

The following are the results on Linux with AcquirePeriod=0.005, 2000 frames

Table 3. Linux system with dual quad-core CPUS (Intel(R) Xeon(R) CPU, E5630 @ 2.53GHz)

Acquire Period	0.02	0.02	0.02	.02	0.02	0.02
# Frames	2000	2000	2000	2000	2000	2000
File plugin	None	TIFF	netCDF single	netCDF stream	HDF5 single	HDF5 stream
Total time	41	42	53	40	58	41
Detector Frames/s	49	48	37.7	50	34.5	49
CPU %	790	820	790	840	720	840
Frames dropped						
Image1	0	0	0	0	0	0
PROC1	1688	1694	1594	1795	1553	1689
ROI1	0	0	0	0	0	0
ROI2	0	0	0	0	0	0
ROI3	0	0	0	0	0	0
ROI4	0	0	0	0	0	0
STATS1	1635	1617	1476	1619	1444	1641
STATS2	1252	1281	1017	1279	914	1258
STATS3	541	566	204	574	188	512
STATS4	480	546	182	557	145	571
STATS5	1746	1738	1662	1743	1627	1748
TIFF1	0	461	0	0		0
NetCDF1	0	0	132	238		0
HDF1	0	0	0	0	109	208

The performance with the netCDF or HDF5 plugins operating in Stream mode is the same as with no file writing plugins enabled. The total time for 2000 images is 41 seconds with no plugins, or with the netCDF or HDF5 plugin in Stream mode. The CPU usage is 800-840%, but the file writers do not slow down the simDetector thread that is computing the images.

However, when writing individual netCDF or HDF5 files it slows down the simDetector by almost 50%, increasing the time for 2000 images from 41 seconds to 58 seconds.

The following are the results on Windows with AcquirePeriod=0.005, 2000 frames

Table 4. Windows 7 64-bit computer system with dual quad-core CPUS (Intel(R) Core(TM) i7-2820QM CPU@ 2.30GHz).

Acquire Period	0.005	0.005	0.005	.005	0.005	0.005
# Frames	2000	2000	2000	2000	2000	2000
File plugin	None	TIFF	netCDF single	netCDF stream	HDF5 single	HDF5 stream
Total time	31.3	188	157	31.3	159	31.4
Detector Frames/s	63.9	10.6	12.7	63.9	12.6	63.7
CPU %	65	20	20	90	25	85
Frames dropped						
Image1	0	0	0	0	0	0
PROC1	1515	320	310	1537	241	1559
ROI1	0	0	0	0	0	0
ROI2	0	0	0	0	0	0
ROI3	0	0	0	0	0	0
ROI4	0	0	0	0	0	0
STATS1	0	0	0	0	0	1185
STATS2	0	0	0	0	0	335
STATS3	0	0	0	0	0	0
STATS4	0	0	0	0	0	0
STATS5	1379	242	181	1407	153	1433
TIFF1	0	418	0	0		0
NetCDF1	0	0	203	1041		0
HDF1	0	0	0	0	0	884

The performance with the netCDF or HDF5 plugins operating in Stream mode is the same as with no file writing plugins enabled. The total time for 2000 images is 31.3 seconds with no plugins, or with the netCDF or HDF5 plugin in Stream mode. The CPU usage is 85-90%, but the file writers do not slow down the simDetector thread that is computing the images.

However, there is clearly a serious problem when individual files are being written. It does not matter if they are TIFF, netCDF, or HDF5. In all cases when writing individual files the total time increases by a factor of 5-6, from 31 seconds to 157-188 seconds.

Conclusion 1: The file writing plugins do not slow down the simDetector thread if they are running in Stream mode, where there is a single file creation for 2000 frames. This is true on Linux and Windows

Conclusion 2: There appears to be a lock problem when files are created, so that if individual files are being written it slows down the simDetector thread. The slowdown is only about 50% on Linux, but is a factor of 5-6 on Windows. Because the simDetector thread is running more slowly the other plugins do not drop as many frames. This problem needs to be investigated and fixed.

Conclusion 3: The throughput of netCDF and HDF5 file writers in Stream mode were as follows:

netCDF, Windows: (2000-1041)/31.3*4MB = 122 MB/s

HDF5, Windows: : (2000-884)/31.4*4MB = 142 MB/s

netCDF, Linux: (2000-238)/40*4MB = 176 MB/s

HDF5, Linux: : (2000-208)/41*4MB = 175 MB/s

The Linux machine is a server with fast disks, the Windows machine is a laptop with relatively slow disk.