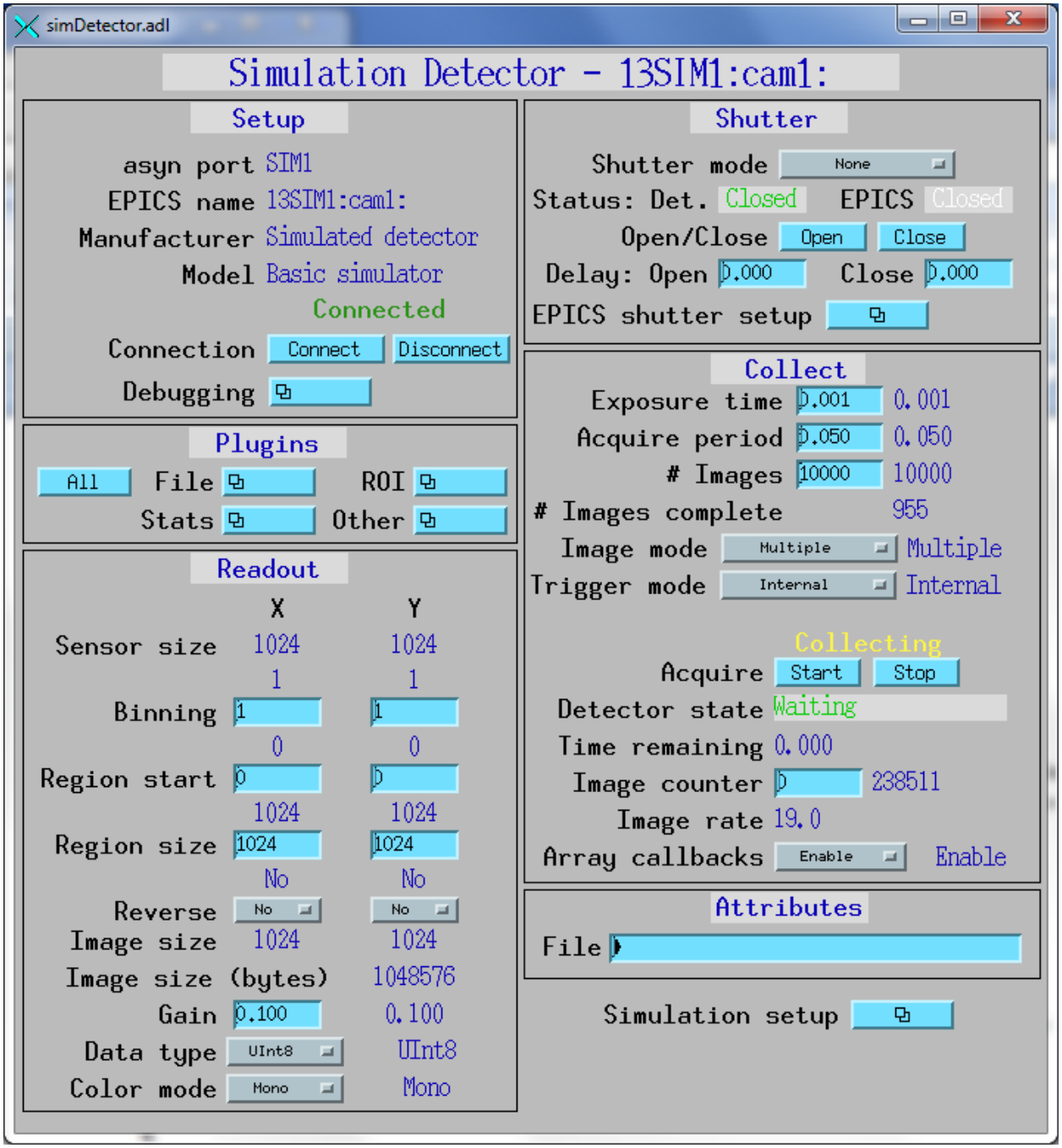
Measurement of Performance of areaDetector Plugin Architecture

Mark Rivers

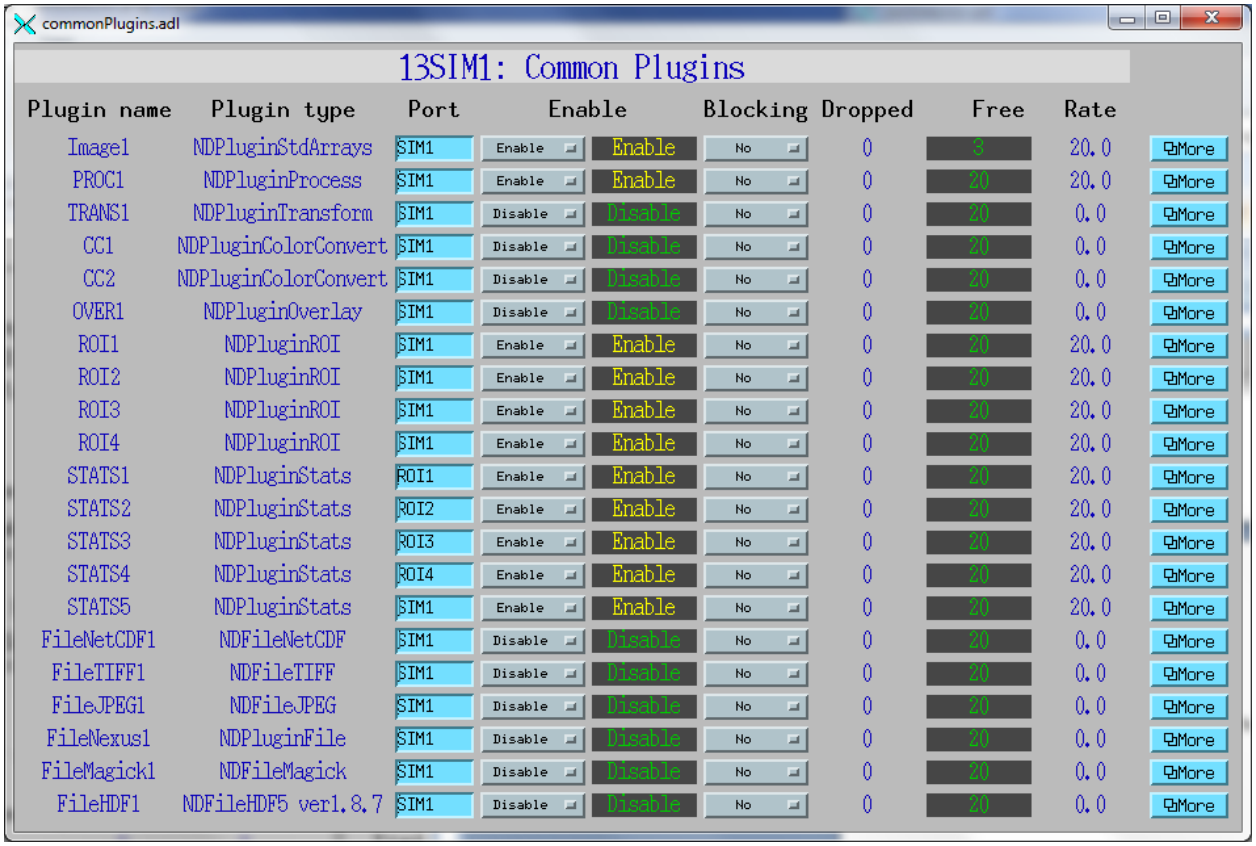
June 21, 2013

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:



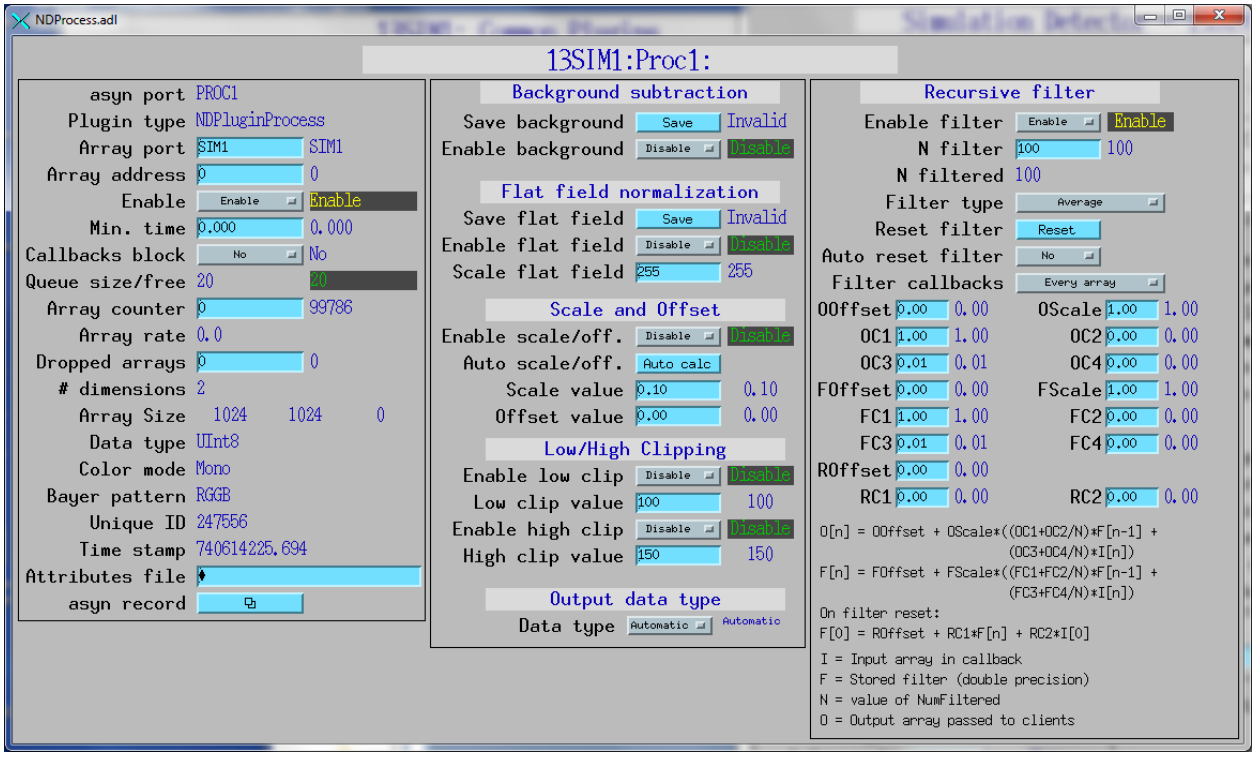
For the first set of measurements the following plugin configuration was used.



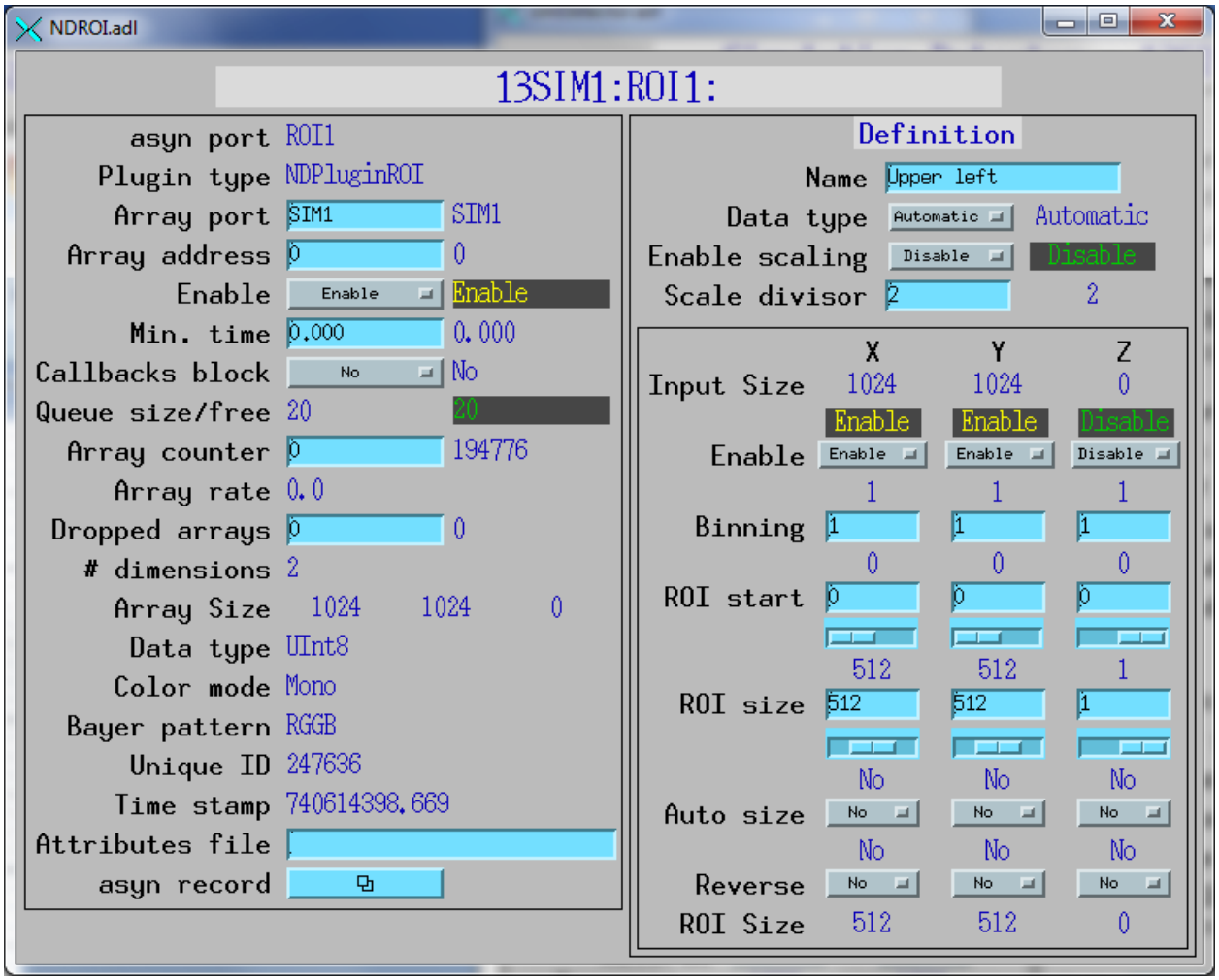
Plugin name	Plugin type	Port	Enable		Blocking	Dropped	Free	Rate	
Image1	NDPluginStdArrays	SIM1	Enable	Enable	No	0	3	20.0	More
PROC1	NDPluginProcess	SIM1	Enable	Enable	No	0	20	20.0	More
TRANS1	NDPluginTransform	SIM1	Disable	Disable	No	0	20	0.0	More
CC1	NDPluginColorConvert	SIM1	Disable	Disable	No	0	20	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.

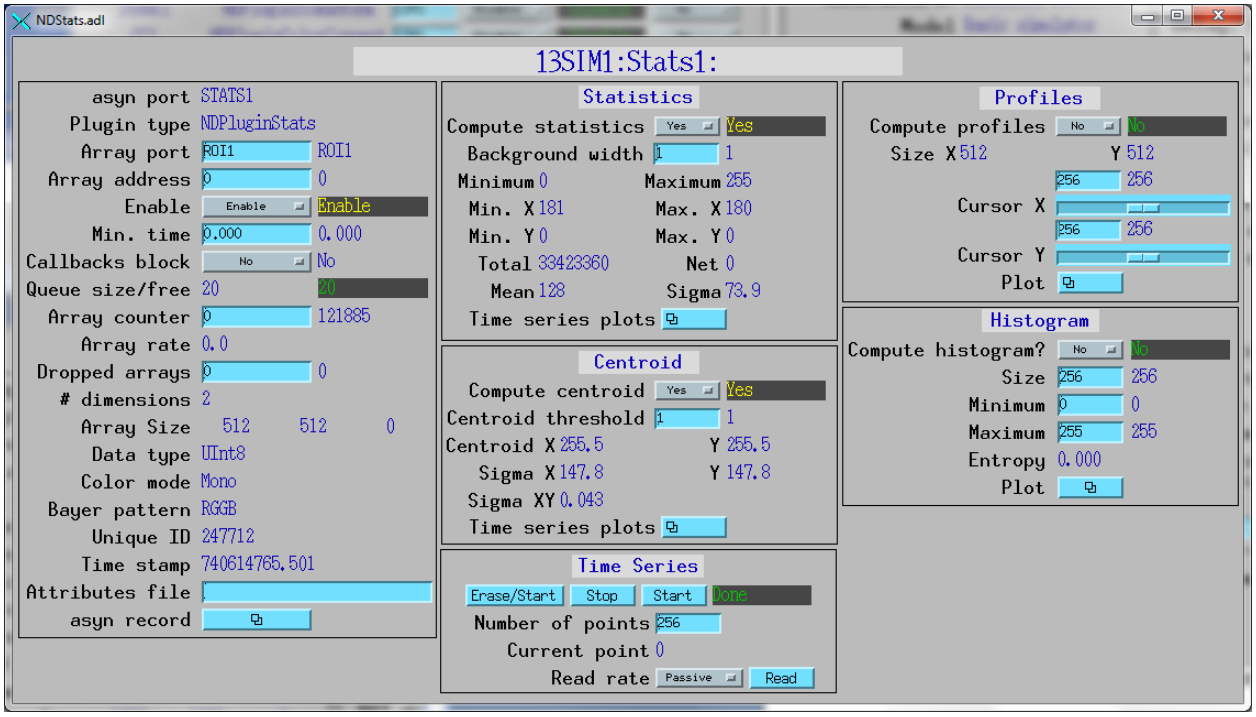
The PROC1 plugin was configured to get its data from the entire detector (SIM1) and to average 100 frames as shown below.



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:



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.



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.