The Impact of the Data Archiving File Format on Scientific Computing and Performance of Image Processing Algorithms in MATLAB Using Large HDF5 and XML Multimodal and Hyperspectral Data Sets

Kelly Bennett[1] and James Robertson[2]
[1]U.S. Army Research Laboratory, Sensors and Electron Devices Directorate, Adelphi, MD
[2]Clearhaven Technologies LLC, Severna Park, MD
U.S.A

1. Introduction

Scientists require the ability to effortlessly share and process data collected and stored on a variety of computer platforms in specialized data storage formats. Experiments often generate large amounts of raw and corrected data and metadata, which describes and characterizes the raw data. Scientific teams and groups develop many formats and tools for internal use for specialized users with particular references and backgrounds. Researchers need a solution for querying, accessing, and analyzing large data sets of heterogeneous data, and demand high interoperability between data and various applications (Shasharina et al., 2007; Shishedjiev et al., 2010).

Debate continues regarding which data format provides the greatest transparency and produces the most reliable data exchange. Currently, Extensible Markup Language (XML) and Hierarchical Data Format 5 (HDF5) formats are two solutions for sharing data. XML is a simple, platform-independent, flexible markup meta-language that provides a format for storing structured data, and is a primary format for data exchange across the Internet (McGrath, 2003). XML data files use Document Type Definitions (DTDs) and XML Schemas to define the data structures and definitions, including data formatting, attributes, and descriptive information about the data. A number of applications exist that use XML-based storage implementations for applications, including radiation and spectral measurements, simulation data of magnetic fields in human tissues, and describing and accessing fusion and plasma physics simulations (Shasharina et al., 2007; Shishedjiev et al., 2010).

HDF5 is a data model, library, and file format for storing and managing data. HDF5 is portable and extensible, allowing applications to evolve in their use of HDF5 (HDF Group). HDF5 files provide the capability for self-documenting storage of scientific data in that the HDF5 data model provides structures that allow the file format to contain data about the file structure and descriptive information about the data contained in the file (Barkstrom, 2001). Similar to XML, numerous applications using the HDF5 storage format exist, such as fusion

and plasma physics, astronomy, medicine and bio-imaging (Shasharina et al., 2007; Dougherty et al., 2009).
In this chapter, we will use hyperspectral images stored in XML and HDF5 format to compare the relative performance of the file format using computationally intensive signal and image processing algorithms running in MATLAB on Windows® 64-bit and Linux 64-bit workstations. Hyperspectral imaging refers to the multidimensional character of the spectral data set, where the acquisition of images takes place over many contiguous spectral bands throughout the visible and infrared (IR) regions (Goetz et al., 1985). Sensor fusion and advanced image processing techniques are now possible using the information from these different bands that allow applications in aerospace, defense, medicine, and other fields of study.
To assist researchers in exchanging the data needed to develop, test, and optimize the techniques, selecting the best file format for computing environments (such as MATLAB) requires additional analysis. Such analysis includes analyzing the relative performance of the file format, including scalability, with respect to various computational tools, computer architectures, and operating systems (Bennett & Robertson, 2010). In this chapter we provide insights into the challenges researchers face with a growing set of data, along with expectations for performance guidelines on workstations for processing large HDF5 and XML hyperspectral image data. Additionally, in this chapter, we provide specific results comparing data load, process, and memory usage for the differing data formats, along with detailed discussions and implications for researchers.

2. Analysis of HDF5 and XML Formats

The goals of this analysis are to:
1. Determine strengths and weaknesses of using HDF5 and XML formats for typical processing techniques associated with large hyperspectral images;
2. Compare and analyze processing times on Windows and Linux 64-bit workstations for HDF5 and XML hyperspectral images; and
3. Identify areas that require additional research to help improve efficiencies associated with processing large HDF5 and XML files, such as hyperspectral images.

3. Methodology for Analysis of HDF5 and XML Formats

To address the analysis goals a set of 100 files containing multimodal hyperspectral images, ranging in size from 57 MB to 191 MB, stored in HDF5 format provided the input for the creation of HDF5 and XML dataset files as part of a preprocessing step for further analysis. The created HDF5 and XML dataset files provided the input to a series of analysis techniques typically associated with image and signal processing. Two different workstations running 64-bit Windows and Linux operating systems are used. The workstations are equipped with MATLAB (scientific programming language). Table 1 displays the descriptions of each of the workstations.
The hyperspectral images were originally stored in HDF5 format and included several different types of metadata in the form of HDF5 Groups and Datasets. Metadata in a typical HDF5 file includes ground truth, frequency bandwidths, raw image data, TIFF (Tagged

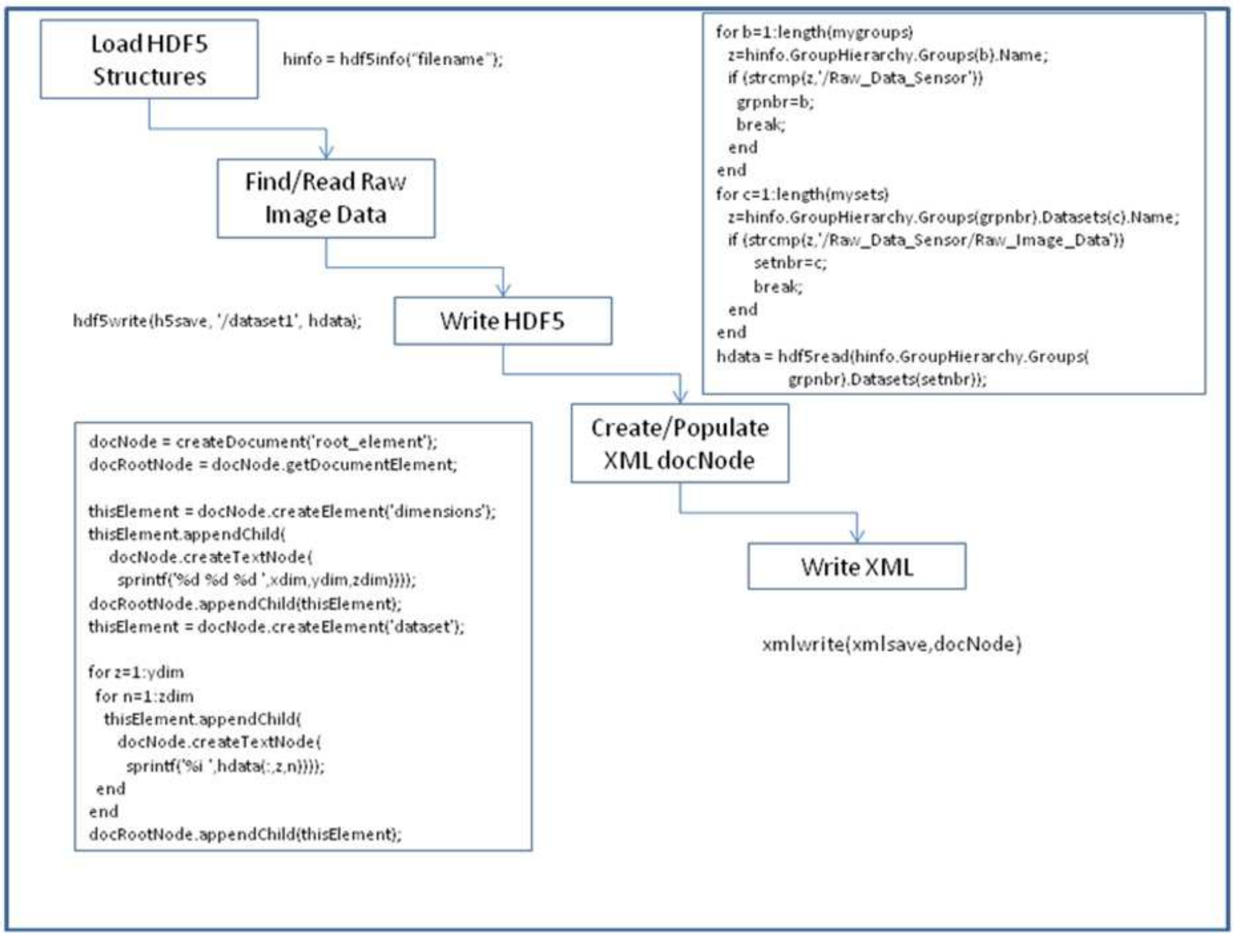
® Registered trademark of Microsoft Corporation.

Image File Format)-formatted images, collection information, and other ancillary information, allowing researchers to understand the images and their collection parameters.

Descriptor	Windows 64-bit	Linux 64-bit
Operating System	Windows 7 home premium	Red Hat Enterprise Linux 5
CPU	Intel i7 920 @2.67 GHz	2 processor, quad core Xeon 2.0 GHz
Memory	6 GB	16 GB
MATLAB version	7.11.0 (R2010b)	7.11.0 (R2010b)

Table 1. Research Equipment Descriptions.

Each original HDF5 file went through a number of preprocessing steps to remove the metadata in preparation for analysis. For analysis purposes, we needed to remove the metadata from the original HDF5 files and create new HDF5 and XML formatted files consisting of only raw sensor data prior to performing image processing. These steps included loading the original HDF5 file structures, searching through the HDF5 groups to find the raw image data, saving the new HDF5 file, creating and populating an XML document node, and saving the XML file. Figure 1 shows the overall steps in processing the original HDF5 file, along with some critical MATLAB code associated with each of those steps.



```
hinfo = hdf5info("filename");
```

```
for b=1:length(mygroups)
  z=hinfo.GroupHierarchy.Groups(b).Name;
  if (strcmp(z,'/Raw_Data_Sensor'))
   grpnbr=b;
   break;
  end
end
for c=1:length(mysets)
  z=hinfo.GroupHierarchy.Groups(grpnbr).Datasets(c).Name;
  if (strcmp(z,'/Raw_Data_Sensor/Raw_Image_Data'))
      setnbr=c;
      break;
  end
end
hdata = hdf5read(hinfo.GroupHierarchy.Groups(
            grpnbr).Datasets(setnbr));
```

```
hdf5write(h5save, '/dataset1', hdata);
```

```
docNode = createDocument('root_element');
docRootNode = docNode.getDocumentElement;

thisElement = docNode.createElement('dimensions');
thisElement.appendChild(
   docNode.createTextNode(
     sprintf('%d %d %d ',xdim,ydim,zdim))));
docRootNode.appendChild(thisElement);
thisElement = docNode.createElement('dataset');

for z=1:ydim
 for n=1:zdim
  thisElement.appendChild(
    docNode.createTextNode(
      sprintf('%i ',hdata(:,z,n))));
 end
end
docRootNode.appendChild(thisElement);
```

```
xmlwrite(xmlsave,docNode)
```

Fig. 1. Original HDF5 File Preprocessing Overview for the creation of HDF5 and XML Dataset Files.

After creating the HDF5 and XML files for the raw sensor data, each file was loaded into MATLAB, converted to an array as needed, and run through a number of image processing steps. XML stores the array data as a large ASCII character string, which requires converting the character array into a numeric array before beginning any processing. Unfortunately, the arrays were too large to use MATLAB's str2num() function, so a novel custom method was developed to read each character and convert it into numbers before writing the numbers into a MATLAB array.

Technique	Description	Example MATLAB Call
Image Adjustment	Maps the values in intensity to new values such that 1% of data saturates at low and high intensities of the original image.	im2= imadjust(im);
Histogram	Calculates a histogram where the variable bin specifies the number of bins used in the histogram.	[COUNTS,X] = imhist(im2,bin);

Descriptive Statistics	Computes the mean and standard deviations of the image values.	imagemean=mean2(im2);
		imagestd=std2(im2);

Median Noise Filter	Performs a median filtering of the image using a 3-by-3 neighborhood.	J=medfilt2(im2);
Weiner Noise Filter	Performs a filtering of the image using pixel-wise adaptive Wiener filtering, using neighborhoods of size fx-by-fy to estimate the local image mean and standard deviation	K = wiener2(im2,[fx fy]);
Sobel Edge Detection	Sobel method finds edges using the Sobel approximation to the derivative. It returns edges at those points where the gradient is maximum	BW1 = edge(im2,'sobel');
Canny Edge Detection	The Canny method finds edges by looking for local maxima of the gradient. The derivative of a Gaussian filter provides the approach for calculating the gradient.	BW2 = edge(im2,'canny');

FFT 2D Threshold Feature Detection	FFT approach to template matching and feature detection.	z= im2(minx:maxx,miny:maxy);
		
		C = real(ifft2(fft2(im2) .* fft2(rot90(z,2),dims(1),dims(2))));
		t=max(C(:)) - .05*max(C(:));

Table 2. Image Processing Technique Descriptions.

Once stored as numeric arrays, the processing for the XML and the HDF5 files were the same and these processing steps include image adjustment, histogram calculation, and descriptive statistics, filtering to remove noise, edge detection and 2-D FFT threshold feature detection. Each of these image-processing techniques includes possible techniques users may invoke when processing hyperspectral images. Table 2 provides a brief description of each of these techniques and an example call within MATLAB. In Table 2, "im" represents the original image and 'im2' represents a processed image of 'im'. Each row in Table 2 shows various processing operations performed on 'im' or 'im2'. Figure 2 shows the flow of the image processing techniques.

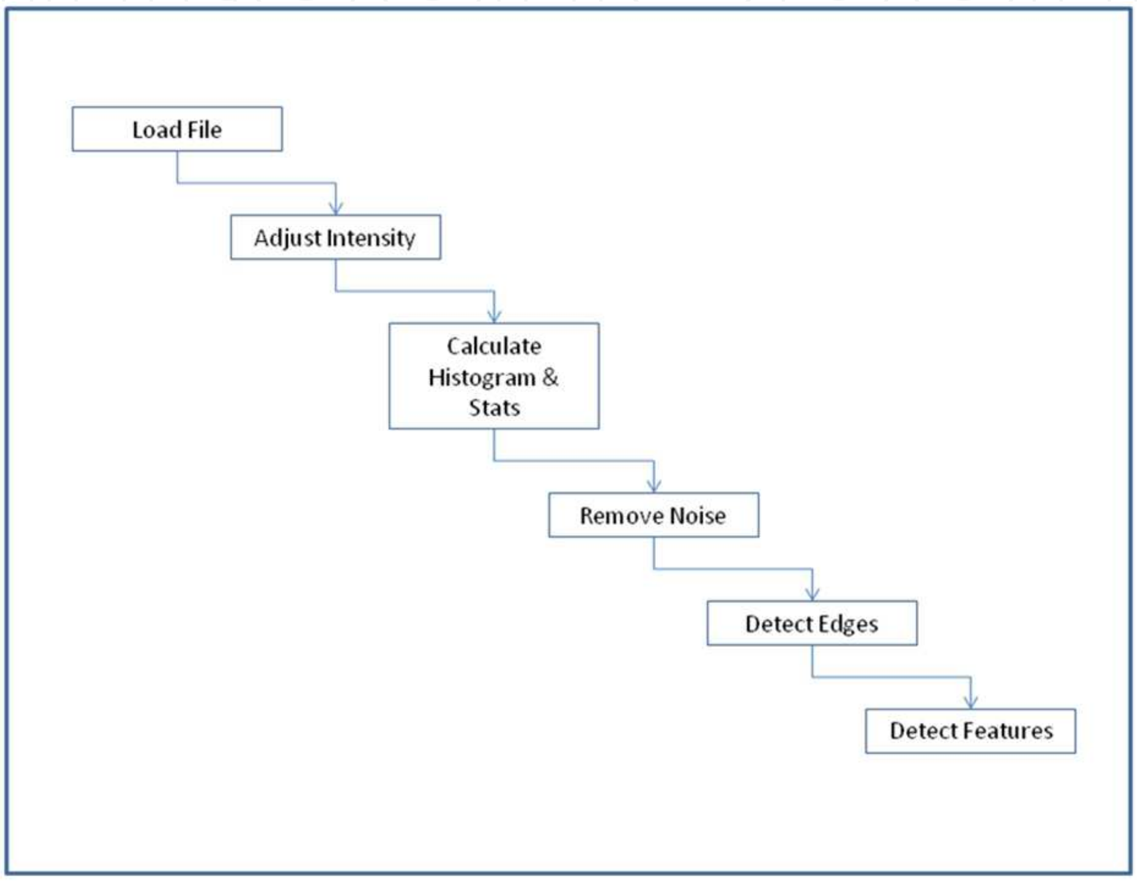


Fig. 2. Image Processing Overview.

Some of the metrics used for assessing the performance of each file format are calculation of load times, process times, and memory usage statistics for each file format and machine. These metrics reveal the computational performance of processing large archived data files in MATLAB using typical image processing algorithms. MATLAB's tic and toc methods were convenient to measure elapsed times associated with each processing step. Hyperspectral images consist of multiple segments representing different spectral bands or narrow frequency bands, with data collected for each image segment and averaged for reporting purposes. For example, an image with 62 segments would generate data for each of the 62 segments and the mean of those values, with the results described in the Results section of this paper. A typical sequence of elapsed time measurement would occur as shown in Figure 3. In the Figure 3 example, all files ("i") and segments ("j") perform the timing process for the image adjustment algorithm.

```
% Adjust the image for better display
tic;
im2= imadjust(im);
adjustIM(i,j)=toc;
```

Fig. 3. Elapsed Time Measurement Example Code.

After loading each created dataset file, both in HDF5 and XML, measuring the memory will determine the average memory usage. For the Windows environment, MATLAB's memory functions perform the process of determining the physical memory available at that point in time. For the Linux environment, system calls to the Linux memory functions determine the physical memory available after loading the file. MATLAB does not provide a Linux memory function at this time. Figure 4 shows a typical Windows memory call.

```
% Measure Windows Memory
[USERVIEW, SYSTEMVIEW] = memory;
pmem(i,j)=SYSTEMVIEW.PhysicalMemory.Available;
```

Fig. 4. Windows Memory Measurement Example Code.

Test	Description	Performance Factors
LU	Perform LU of a full matrix	Floating-point, regular memory access
FFT	Perform FFT of a full vector	Floating-point, irregular memory access
ODE	Solve van der Pol equation with ODE45	Data structures and MATLAB function files
Sparse	Solve a symmetric sparse linear system	Mixed integer and floating-point
2-D	Plot Bernstein polynomial graph	2-D line drawing graphics
3-D	Display animated L-shape membrane logo	3-D animated OpenGL graphics

Table 3. MATLAB's Benchmark Descriptions.

Prior to running the algorithms, each computer system performed baseline benchmarks. MATLAB has a convenient built-in benchmark named "bench" that executes six different MATLAB tasks and compares the execution speed with the speed of several other computers. Table 3 shows the six different tasks.
The LU test performs a matrix factorization, which expresses a matrix as the product of two triangular matrices. One of the matrices is a permutation of a lower triangular matrix and the other an upper triangular matrix. The fast Fourier transform (FFT) test performs the discrete Fourier transform computed using an FFT algorithm. The ordinary differential equation (ODE) test solves equations using the ODE45 solver. The Sparse test converts a

matrix to sparse form by removing any zero elements. Finally, the 2-D and 3-D measure 2-D and 3-D graphics performance, including software or hardware support for OpenGL (Open Graphics Library).

The benchmark results in a speed comparison between the current machine and industry-available machines.

4. Data analysis

Data analysis included calculating descriptive statistics on each test to include mean, standard deviation, variance, minimum and maximum values, and t-test analysis; to determine relationships and differences in performance measurements comparing XML and HDF5 formats for both computer systems. The t-test is one of the most commonly used statistics to determine whether two datasets are significantly different from one another (Gay & Airasian, 2003). The t-test determines if the observed variation between the two datasets is sufficiently larger than a difference expected purely by chance. For this research, the significance level (α) was set at 0.05. This value is commonly accepted and is the default value for many statistical packages that include the t-test (Gay & Airasian, 2003; SAS Institute, 2003; MathWorks, 2011).

For each processing, memory, or loading algorithm, the descriptive statistics for each hyperspectral image create relevant data for a final analysis. The information obtained from averaging across each segment of the multiple segmented images creates the analytical data products used in the results.

In addition to the descriptive statistics for each process, graphical plots illustrate the load times, process times, and memory usage as a function of file size for each data type and test environment. These plots provide an ability to identify differences between the XML and HDF5 data types and possible processing bottlenecks and limitations.

5. Results and Implications

Scientists and researchers need a reliable format for exchanging large datasets for use in computational environments (such as MATLAB). MATLAB has many advantages over conventional languages (such as FORTRAN, and C++) for scientific data analysis, such as ease of use, platform independence, device-independent plotting, graphical user interface, and the MATLAB compiler (Chapman, 2008). Previous results have shown HDF5 format provided faster load and process times than XML formats, and loads large amounts of data without running into memory issues (Bennett & Robertson, 2010). This research supports these findings.

This section provides results and discussion of this current research. After the baseline benchmarks provide results for each machine, the analysis will show example images and descriptive statistics for each image-processing algorithm, along with tables, plots, and discussion comparing HDF5 and XML formats for each task.

Table 4 shows the average of 10 MATLAB bench time results for each of the machines for the LU, FFT, ODE, Sparse, 2D, and 3D tests. For most tests, the Windows 64-bit machine performed better (as indicated by smaller execution times) than the Linux 64-bit machine. One exception to this was the 2D graphics, where the Linux 64-bit machine was slightly faster than the Windows machine. Based on these results, the Windows 64-bit machine should perform slightly faster for the subsequent image processing tasks.

Machine	LU	FFT	ODE	Sparse	2D	3D
Windows 64-bit	0.0389	0.0511	0.1353	0.1789	0.4409	0.7531
Linux 64-bit	0.0872	0.1221	0.2267	0.3137	0.3301	0.8154

Table 4. MATLAB's Bench Results.

Figure 5 shows a typical image used in this analysis. This image represents one specific frequency range (spectral band) for a 460 x 256 image after adjusting of the intensity for display.

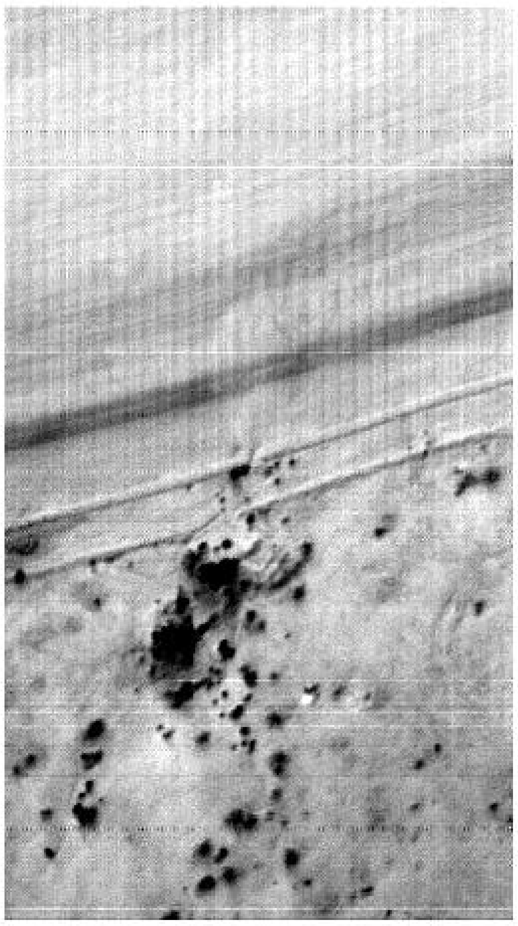


Fig. 5. Example 460 x 256 Image.

A quad chart (Figure 6) displays processed images showing some of the techniques. The first image in Figure 6 is an image in the upper-left corner representing the image adjusted for intensity. The image in the upper-right corner represents the image after the Weiner noise filter is applied. Next, the image in the lower-left corner represents the image after the Canny edge detection is applied. Lastly, the image in the lower-right corner represents the FFT threshold results.
Recall from Figure 1, preparing the images for processing requires several steps. The first step was to load the HDF5 structures, followed by finding and loading the HDF5 raw image data, saving the HDF5 raw image data, populating the XML docNode, and saving the XML raw image data.

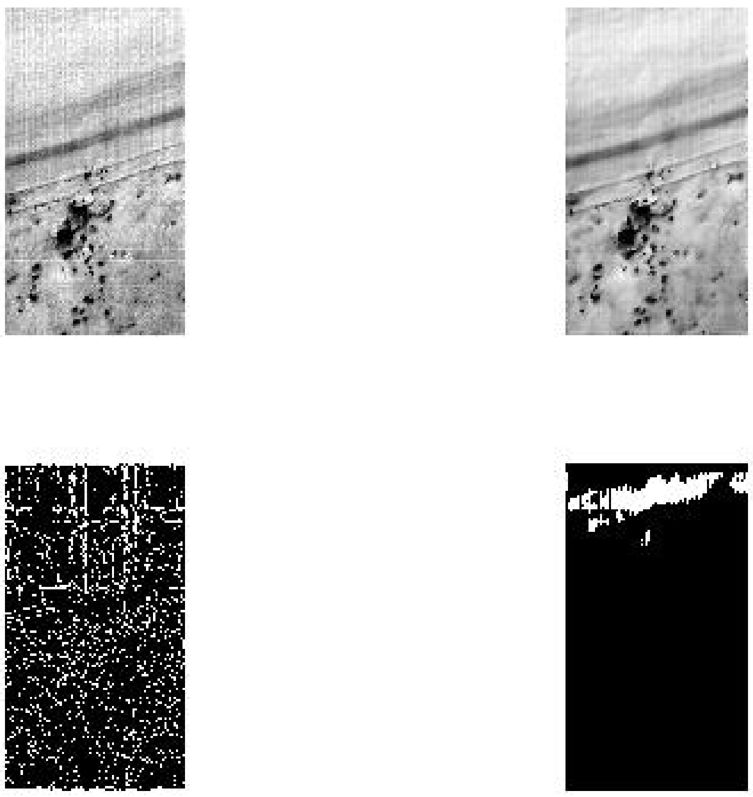


Fig. 6. Quad Image Example 460 by 256 Image.

A total of 100 original HDF5 files, ranging from 57 to 191 MB in size, provide the input for the creation of the HDF5 and XML dataset files. Table 5 displays the original HDF5 file size statistics for this research. The original HDF5 files contained ground truth, collection information, processed data, and spectral content, in addition to the raw image data. The computed image processing statistics use only the raw image data extracted from the HDF5 files and saved to HDF5 and XML formats.

Descriptor	Value (MB)
Mean	116.7304
Minimum	56.9807
Maximum	191.7729
Standard Deviation	28.5472
Variance	814.9406

Table 5. Original HDF5 File Size Descriptions.

The average times associated with each of these steps are shown in Table 6 for the Windows 64-bit and Table 7 for Linux 64-bit machine. The column labeled **"Total (s)"** represents the sum of each of the processing steps for the respective machines. For the current configuration of the Windows 64-bit machine, the mean preparation time per file was just over 9 s, with preparation times ranging between almost 7 and approximately 16.5 s. For the current configuration of the Linux 64-bit machine, the mean preparation time per file was almost 11 s, with times ranging between almost 9 and approximately 19.5 s.

Statistic	Load (s)	Read (s)	HDF5 Save (s)	docNode (s)	XML Save (s)	Total (s)
Mean	0.3313	0.3665	0.0996	7.129	1.1577	9.0841
Minimum	0.0261	0.0097	0.0168	6.0196	0.9191	6.9913
Maximum	1.1602	0.7236	0.5047	12.2276	1.9504	16.567
Standard Deviation	0.2269	0.1384	0.1275	1.8607	0.301	
Variance	0.0515	0.0192	0.0163	3.4623	0.0906	

Table 6. Windows 64-bit HDF5 Data Average Preparation Times.

Statistic	Load (s)	Read (s)	HDF5 Save (s)	docNode (s)	XML Save (s)	Total (s)
Mean	0.043	0.0226	0.0653	9.5711	1.0478	10.75
Minimum	0.0076	0.0107	0.044	8.146	0.7723	8.9806
Maximum	0.3131	0.1311	0.1404	16.775	2.1607	19.52
Standard Deviation	0.0382	0.0171	0.0245	2.6023	0.3637	
Variance	0.0015	0.0003	0.0006	6.7721	0.1322	

Table 7. Linux 64-bit HDF5 Data Average Preparation Times.

Table 8 shows the average free physical memory for each system during the preprocessing steps. Free physical memory can vary throughout a run based on system processing during the run and the amount of memory allocated to MATLAB for processing the run. For all runs during this research, the MATLAB Java heap memory was set to its maximum possible value to avoid any potential out-of-memory issues. In MATLAB version 2010b, selecting File, then Preferences, then General, and then Java Heap Memory, and then using the scroll bar to set its maximum setting changes the memory. The maximum setting for the Windows 64-bit machine was 1533 MB, while the maximum setting for the Linux 64-bit machine was 4011 MB. One trade-off with the Java heap memory being larger in Linux is that less physical memory is available for the run. However, increasing the Java heap memory does allow for larger possible Java objects, which is useful when dealing with large image arrays.

Statistic	Windows 64-bit (GB)	Linux 64-bit (GB)
Mean	2.7223	0.2374
Minimum	2.5234	0.0791
Maximum	2.9228	1.8715
Standard Deviation	0.0738	0.3696
Variance	0.0054	0.1366

Table 8. Free Physical Memory during HDF5 Preparation Steps.

After the preparation steps are complete, saving the raw image data to HDF5 and XML files is the next step. The new raw image files in HDF5 and XML contain only the image dimension information and the raw image pixel values. Table 9 provides the file statistics of the raw image data in both HDF5 and XML format. In all cases, the XML files are larger compared to the HDF5 files. In most cases, the resulting XML file is between 2.5 and 3 three times as large as the similar HDF5 file. This finding is consistent with other published results (Bennett & Robertson, 2010).

Statistic	HDF5 Raw Image Size (MB)	XML Raw Image Size (MB)
Mean	17.8374	49.3223
Minimum	13.9301	41.0791
Maximum	30.1252	86.0207
Standard Deviation	6.6819	13.7911
Variance	44.6480	190.1954

Table 9. HDF5 and XML Raw Image File Size Statistics.

After saving the raw image data to HDF5 and XML files, each file was loaded and processed according to the steps shown previously in Figure 2. These steps include loading the file, adjusting the image, calculating image statistics, removing noise, detecting edges, and detecting features. Algorithms include two different noise removal algorithms (Median and Weiner Filtering) and two different edge detection algorithms (Sobel and Canny). All of these algorithms, unmodified for this research effort, are available within the MATLAB Image Processing toolbox.

Table 10 shows the statistical results of the execution times for each of these image-processing algorithms for HDF5 and XML formats for the Windows 64-bit. Table 11 shows the results for the Linux 64-bit machine.

	HDF5					**XML**				
Process	Mean	Min	Max	Std	Var	Mean	Min	Max	Std	Var
Load (s)	0.0258	0.0013	0.0530	0.0072	0.0001	0.7309	0.6117	1.3135	0.2059	0.0424
Adjust (s)	0.0162	0.0058	0.0778	0.0235	0.0006	0.0419	0.0055	0.2366	0.0828	0.0069
Histogram (s)	0.0128	0.0013	0.0730	0.0262	0.0007	0.0225	0.0022	0.1256	0.0447	0.0020
Mean2 (s)	0.0005	0.0001	0.0026	0.0008	0.0000	0.0007	0.0001	0.0037	0.0013	0.0000
STD2 (s)	0.0124	0.0009	0.0735	0.0261	0.0007	0.0089	0.0005	0.0489	0.0179	0.0003
Median (s)	0.0057	0.0021	0.0273	0.0081	0.0001	0.0822	0.0097	0.4280	0.1545	0.0239
Weiner (s)	0.1111	0.0098	0.6541	0.2309	0.0533	0.1307	0.0088	0.7117	0.2597	0.0674
Sobel (s)	0.0663	0.0069	0.3890	0.1347	0.0181	0.0661	0.0051	0.3542	0.1288	0.0166
Canny (s)	0.7276	0.0673	4.1964	1.4975	2.2425	0.5622	0.0532	2.9744	1.0781	1.1622
FFT Feature (s)	0.1222	0.0124	0.6884	0.2398	0.0575	0.1461	0.0122	0.7627	0.2759	0.0761
Total (s)	**1.1006**	**0.1079**	**6.2351**			**1.7922**	**0.7090**	**6.9594**		

Table 10. Windows 64-bit HDF5 and XML Image Processing Execution Times.

On both the Windows and Linux machines, the total execution times for the HDF5 files were significantly less than the total execution times for the XML files. Comparing the results for the mean execution time for the Windows machine, HDF5 demonstrates excellent performance (~1.1 s) compared to XML (~1.8 s). The execution times for the windows machine ranged between ~0.1 and ~6.2 s for the HDF5 files, compared to ~0.7 – ~6.9 s for the XML files. Similarly, comparing the results for the mean execution time for the Linux machine, HDF5 demonstrates excellent performance (~1.5 s) compared to XML (~3.1 s). The execution times for the Linux machine ranged between ~0.15 and ~9.2 s for the HDF5 files, compared to ~1.3 – ~12.3 s for the XML files.

The total execution time difference for both the Windows and Linux machines is primarily due to the "load" process. Loading XML files requires far more execution time due to the larger file sizes of the created XML data files (~3 times larger file size when storing the raw data in XML format).

Additional loading difficulties with XML files include:

1. Slowness of the serialization process of converting Unicode XML into binary memory storage (McGrath, 2003).
2. MATLAB loading algorithm ('xmlread' method) uses the Document Object Model (DOM) to load XML files. DOM is memory and resource intensive, and can consume as much as 10 times the computer memory as the size of the actual XML data file (Wang et al., 2007).
3. In general, and of particular concern for users performing 32-bit processing, processing speeds associated with XML loading can be greatly diminished as virtual memory becomes insufficient compared with the size of the XML file as the computer starts to run out of memory.

	HDF5					**XML**				
Process	Mean	Min	Max	Std	Var	Mean	Min	Max	Std	Var
Load (s)	0.0199	0.0021	0.0509	0.0095	0.0001	1.3520	1.2015	2.0415	0.2444	0.0597
Adjust (s)	0.0250	0.0087	0.1276	0.0371	0.0014	0.0757	0.0095	0.4293	0.1503	0.0226
Histogram (s)	0.0160	0.0017	0.0914	0.0326	0.0011	0.0290	0.0028	0.1697	0.0598	0.0036
Mean2 (s)	0.0007	0.0003	0.0036	0.0010	0.0000	0.0020	0.0004	0.0106	0.0036	0.0000
STD2 (s)	0.0216	0.0019	0.1309	0.0447	0.002	0.0136	0.0012	0.0806	0.0285	0.0008
Median (s)	0.0631	0.0074	0.3595	0.1265	0.0160	0.1135	0.0127	0.6524	0.2291	0.0525
Weiner (s)	0.1564	0.0179	0.8985	0.3137	0.0984	0.1615	0.0140	0.9496	0.3351	0.1123
Sobel (s)	0.0699	0.0090	0.4486	0.1385	0.0192	0.1221	0.0095	0.7272	0.2573	0.0662
Canny (s)	0.8779	0.0746	5.1806	1.8290	3.3453	0.9195	0.0667	5.4927	1.9447	3.7817
FFT Feature (s)	0.2810	0.0229	1.8790	0.5845	0.3416	0.2960	0.0235	1.7437	0.6157	0.3791
Total (s)	**1.5315**	**0.1465**	**9.1706**			**3.0850**	**1.3417**	**12.2973**		

Table 11. Linux 64-bit HDF5 and XML Image Processing Execution Times.

Some other results worth mentioning confirm the expected relative calculation times between differing noise filters and edge detection methods. As expected, the Weiner Filter (using adaptive techniques) took more time than the Median Filter. In addition, the more complex Canny edge detection algorithm took more time than the Sobel edge detection algorithm.

The load times were larger for the XML files compared to the HDF5 files. This difference is most likely due to the larger XML file size. Figure 7 visually displays the load times for the XML and HDF5 files for the Linux 64-bit machine. Figure 8 shows a similar result for the Windows 64-bit machine.

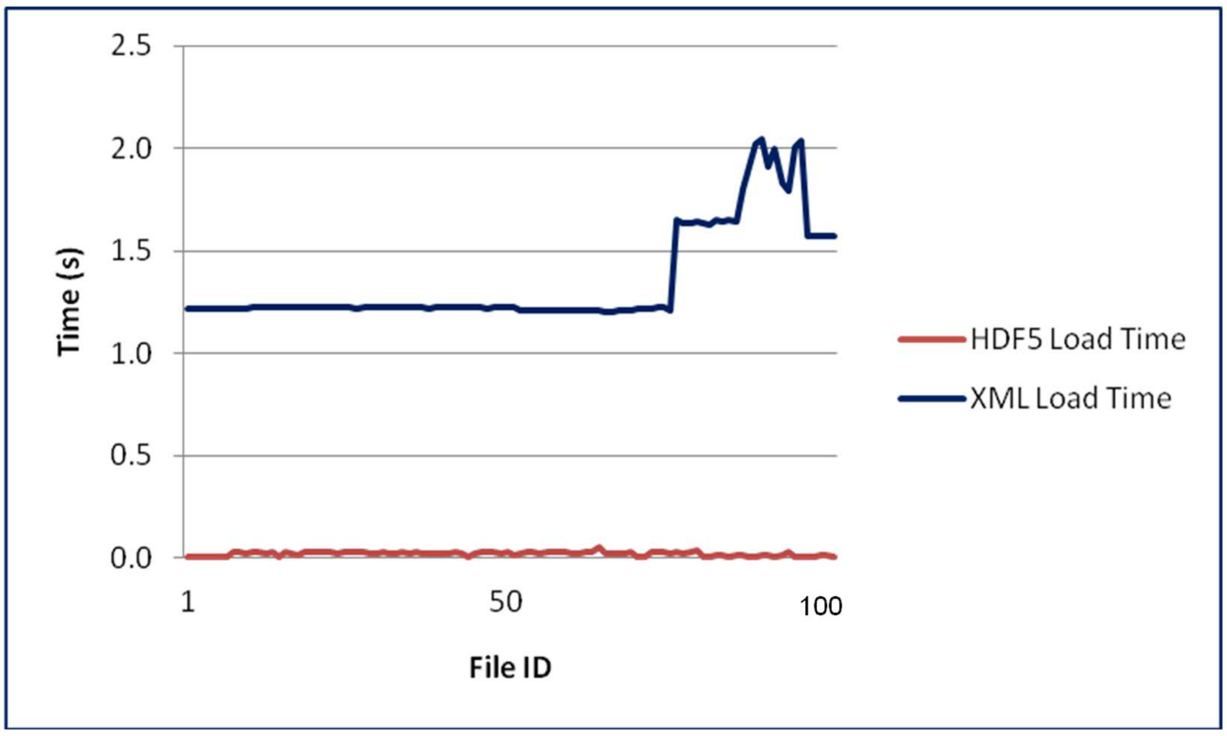


Fig. 7. Linux 64-bit XML and HDF5 Load Times.

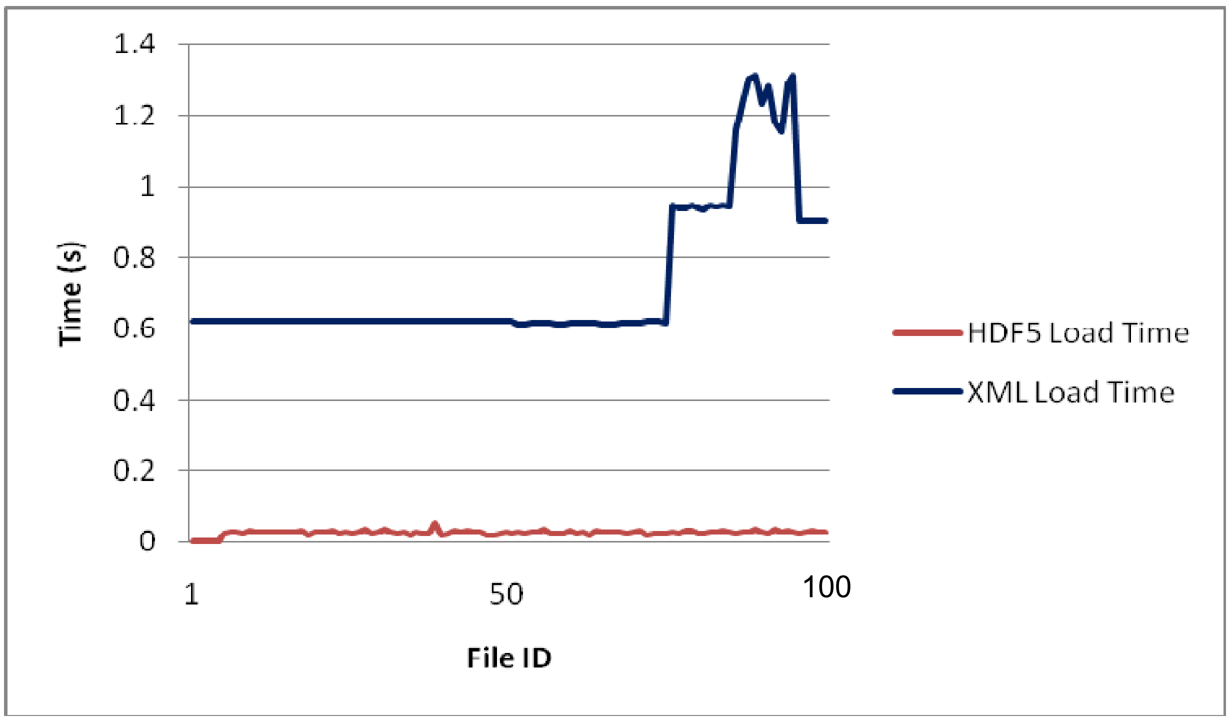


Fig. 8. Windows 64-bit XML and HDF5 Load Times.

Corresponding increases in XML file size contribute to the large jumps observed in the XML load times around file ID 75 and 90 (Figures 7 and 8). Similar arguments made earlier in the chapter (slowness of serialization of converting Unicode to binary storage and resource intensive DOM loading methods) offer explanation of the larger loading process times compared to the more efficient loading of HDF5 binary files.

HDF5 load times do not significantly vary depending on file size. Efficient methods of loading HDF5 binary data files, combined with excellent memory usage and use of computing resources, into the MATLAB workspace, demonstrate the superior benefit of archiving data in HDF5 versus XML. HDF5 provides seamless integration of data into MATLAB without performance degradation (especially for large data sets) and is the 'de facto' standard for MATLAB data files containing workspaces over 2 GB in size (Mather & Rogers, 2007).

The load times (Figures 7 and 8) for both HDF5 and XML show similar behavior on both the Windows and Linux machines. The cross platform behavior demonstrates the file size dependency for XML loading performance, and the lack of file size dependency for HDF5 loading performance. As expected from the benchmark testing results, the XML loading performance on the Windows machine is slightly faster than the Linux.

An additional processing step is required to prepare the large raw data for processing. In XML files, the raw image data is stored as ASCII characters with whitespace separators. As the image gets larger, converting from the ASCII character data to a MATLAB array can take considerable time. MATLAB has a num2str() function that works very nice for small arrays, but this function would not work for these large character arrays. A novel process allows the reading of each character, one at a time, parse on spaces, and then load into the array, resulting in a tremendous savings (as much as two orders of magnitude) in processing time. C or other software development languages may provide other more efficient methods to reduce this processing restriction. However, preparing the XML data for processing is a very important process step. Additional new research and software tools may simplify and expedite the process.

T-test analysis on the total image processing times confirmed that there was a significant difference between the HDF5 and XML file processing times not attributable to random chance. Specifically, HDF5 files took less processing time than XML files on the Windows 64-bit machine ($t\ (198) = 2.27$, $\rho = .0014$) and the Linux 64-bit machine ($t\ (198) = 3.25$, $\rho = .0244$). The t (198), or t-value, represents the difference of the mean values for total processing times for HDF5 and XML, respectively, divided by the standard error of the two means. The 198 represents the degrees of freedom, or sample size minus 2 for an unpaired t-test, which is appropriate for the independent groups in this analysis. The important value (ρ) represents the probability of the difference (t-value) being due to chance is .0014 for the Windows 64-bit machine, and .0244 for the Linux 64-bit machine. Setting the significance level to .05 indicates that in both cases, the difference in processing times between HDF5 and XML is not by chance. These results suggest a significant difference between the total process times for HDF5 and XML files for both machines. Further t-test analysis on the individual components contributing to the total process time indicated significant differences in execution times for load, adjust, and mean calculations for the Linux 64-bit machine and load, adjust, and median noise filter for the Windows 64-bit machine. It seems reasonable the load times would be different between the XML and HDF5 formats. To provide insight into the differences between the XML and HDF5 formats for the image

adjust, median noise filter, and image mean calculations, requires additional research and analysis, since these routines should provide similar results because the data format should not impact the results for these processes.
Table 12 displays the t-test results for each of the components, resulting in significant differences between the XML and HDF5 files. The t-test results for each of the other components shows no significant difference between XML and HDF files.

Test	Windows 64-bit	Linux 64-bit
Load	t (198) = 34.39, ρ < .0001	t (198) = 54.73, ρ < .0001
Image Adjust	t (198) = 2.99, ρ = .0031	t (198) = 3.29, ρ = .0012
Image Mean	t (198) = 1.55, ρ = .122 (Not significant)	t (198) = 3.35, ρ = .0009
Median Filter	t (198) = 4.97, ρ < .0001	t (198) = 1.93, ρ = .050 (Not significant)

Table 12. T-test Process Time Components Results.

Figures 9 and 10 graphically depict these findings by displaying the total processing time for the HDF5 and XML files for the Linux 64-bit and Windows 64-bit test systems. In both cases, the XML process times were significantly greater than the HDF5 process times.

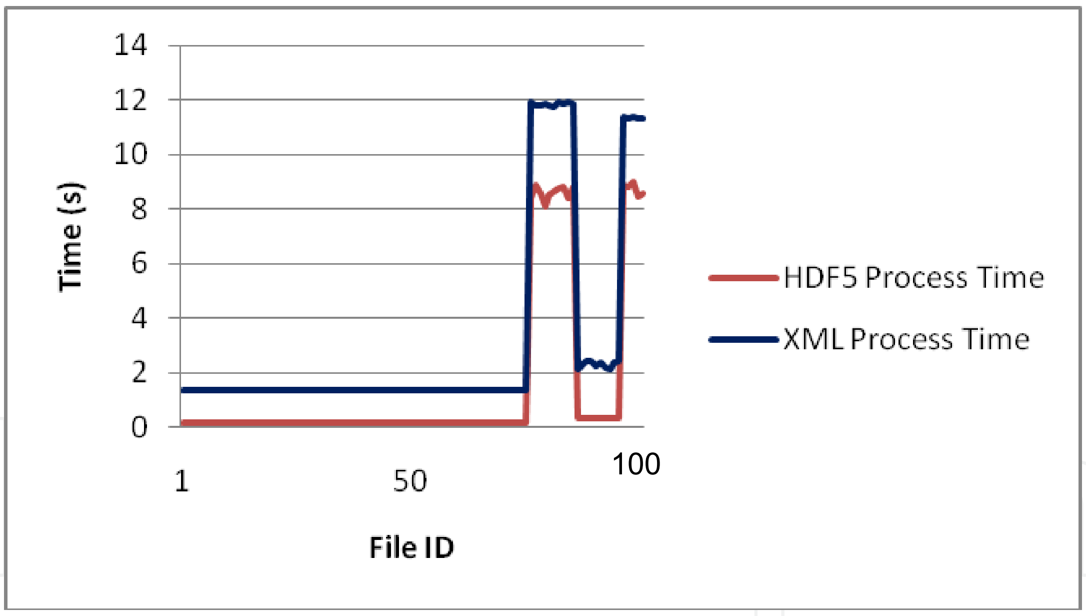


Fig. 9. Linux 64-bit Image Processing Times- HDF5 and XML.

For each file format and test machine, the amount of calculated free physical memory usage during the image processing stage shows definite differences between the file formats. Table 13 shows the descriptive statistics of these data. Similar to the preprocessing step, setting the maximum Java heap memory to maximum for each run results in no out-of-memory errors. For both machines, the XML files required more physical memory than the HDF5 files, as indicated by less free physical memory in Table 13. This result is consistent with XML loading requiring relatively large amounts of memory compared to the XML file size (Wang et al., 2007).

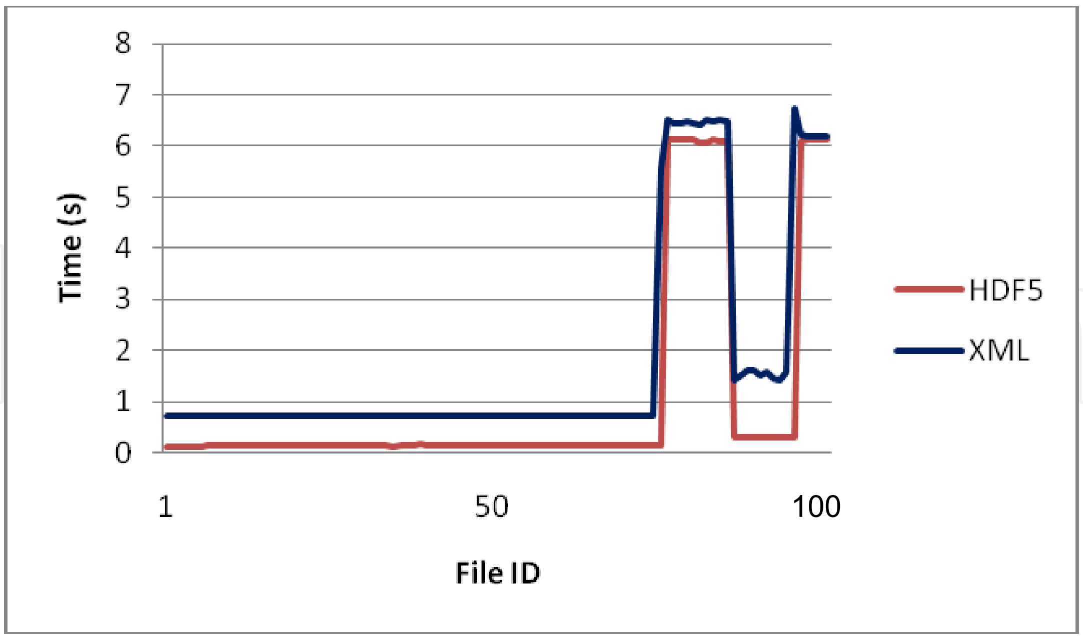


Fig. 10. Windows 64-bit Image Processing Times- HDF5 and XML.

Statistic	Windows 64-bit (GB)		Linux 64-bit (GB)	
	HDF5	XML	HDF5	XML
Mean	3.6608	3.2726	13.6388	11.0808
Minimum	3.5995	3.0064	11.9786	11.0215
Maximum	3.6874	3.7858	14.4033	11.1872
Standard Deviation	0.0194	0.4446	0.6277	0.0924
Variance	0.0004	0.1976	0.394	0.0085

Table 13. Free Physical Memory during Image Processing Steps.

6. Ethics of data sharing

There is a large, complex set of concerns when openly sharing data — especially electronic data over the Internet. From a scientific viewpoint of discovery, open sharing of scientific data allows many researchers and scientists the ability to form a common understanding of data, which is essential for furthering science. However, there are many ethical concerns in the process of sharing data, particularly over the Web. For example, a given medical study group collects sensitive, personal medical information as part of a medical case study using public government funds. All of the data is stored (archived) on a government computer system. Many years later, another researcher wants to use the data for another study, which could help save the lives of many people. Should the second researcher be able to use the archived data for a purpose other than the intent of the original study? Many arguments come into discussion in this situation. The right to use data paid for with publically collected funds seems reasonable; however, what about the right of human participants to privacy? What happens if a data release into the public domain harms any of the participants? Such harm may take the form of job loss or denial of life insurance, etc. The ethics of sharing data is complex and the ethical dilemma of sharing data is an area of study requiring much thought and discussion.

Many of the ethical concerns stem from a balance of beneficial results from sharing data versus ethical concerns researchers have in such sharing. Ethical data sharing and management involves reconciliation of diverse conflicting values (Sieber, 2005). Among these concerns are the sharing of data for the benefit of society and science, while protecting the interest of human participants in data collections (Mauthner & Perry, 2010). For many years, researchers took the position of protecting the interests of the human participants in such data; however, with the advent of sharing data across the Web, the interest of human participants is certainly less sure and threatens the overall fabric of the trust-based relationship that exists between researcher and participant. A definite loss of data control can exist when sharing data across the Web, possibility resulting in the loss of privacy and protection of human participants (Mauthner & Perry, 2010).

Another ethical concern is the rights of those who collect data and receive no recognition by those who download the data through public Web interfaces for use in their research. The process of collecting high quality data requires much time, effort, and expense; moreover, many of the individuals who collect data (data producers) are in a positional or career situation where they are vulnerable to receiving little recognition for their data collection efforts by indiscriminate availability of data over the Web. Such individuals are not nearly as protected as data users, such as algorithm designers, who can protect their interests through intellectual property rights (Mauthner & Perry, 2010).

Along with recognition of the data as a contribution deserving recognition, intellectual property rights assigning ownership and rightful claims to the data are another ethical concern. Reductions or even elimination of researcher's data rights occurs when funding agencies require a researcher to share data, especially over the Web, allowing anyone to access the information. Certain government agencies are always balancing the public's right to information collected with public funds, and the right to protect both the researcher's intellectual property and the test participant's privacy rights.

Archiving and disseminating data over the Web creates a "data as commodity" mindset, where the ethical concerns of both the researcher and human participant become lost in the impersonal downloading of archived data (Mauthner & Perry, 2010). When sharing data, regardless of the methods, confidentially of human participants is important at all times. Data providers must take great care in judging the sensitivity of the data and may find it necessary to restrict access based on ethical, legal, or security justifications, even in the case of publicly funded data collections. Further safeguards in data dissemination include restricting others (end users) of disseminating data as a third party; thus, requiring an end user to go to the original source to acquire the data (MIT Libraries, 2011).

The ethics of data sharing is clearly more complex today than before the advent of the Internet. However, many general guiding principles apply to all data sharing situations. As a core group of guiding principles, every data collector and provider has a duty to:

1. Protect the confidentially of human participants in data collections (UK Data Archive, 2011).
2. Avoid providing sensitive information of human test participants, which may endanger data test participants (UK Data Archive, 2011).
3. Consult with the test participants on making data publically available and be sensitive to their wishes (UK Data Archive, 2011).
4. Inform the test participants on the use of the data, and the methods, procedures, and intentions of archiving and disseminating the data, prior to using them as test participants (UK Data Archive, 2011).

5. Make data available to the public, which doesn't violate ethical, legal, or security principles (UK Data Archive, 2011).

7. Conclusions

This research processed 100 large hyperspectral images in both HDF5 and XML formats on Windows 64-bit and Linux 64-bit machines. A number of image processing steps available within MATLAB, including intensity adjustment, histogram calculation, statistical analysis, noise removal, edge detection and feature extraction, provided the algorithms to fulfill the goals of the research:

1. Determine strengths and weaknesses of using HDF5 and XML formats for typical processing techniques associated with large hyperspectral images.
2. Compare and analyze processing times on Windows and Linux 64-bit machines for HDF5 and XML hyperspectral images.
3. Identify areas that require additional research to help improve efficiencies associated with processing large HDF5 and XML hyperspectral images.

The research identified a number of strengths and weaknesses. First, the overall image processing results show reduced processing times for images stored in HDF5 compared to XML format. The main contribution to this difference is the large load time and the preprocessing step required to convert an ASCII XML character string to a numeric array in MATLAB. The relative size of the files is the main factor in the difference in load speed with the XML files being almost three times as large as the HDF5 files. A larger file will always take more time to load using any application.

The preprocessing required to convert an ASCII XML character string to a numeric array was very time-consuming and a potential huge process bottleneck. The processing of large XML files requires additional tools and approaches with an easier out-of-the-box solution, making XML processing more practical. In addition to the processing time differences, HDF5 requires less physical memory and, hence, allows larger objects to be loaded without out-of-memory errors. HDF5 data files are much smaller (~3 times) than the corresponding XML versions of same data files. Binary files in general are far more efficient in storing numerical data than XML files using Unicode. As discussed earlier, XML loading of data can consume as much as 10 times the amount of computer memory as the size of the actual XML file (Wang et al., 2007), and conversion of Unicode to binary storage is memory intensive requiring much more physical memory and resources than the loading of HDF5 files of similar data (McGrath, 2003). MATLAB can process very large arrays, but it will run out of memory quickly when processing very large XML files. On another test machine that was running MATLAB with only 760 MB Java heap memory, several of the larger XML datasets would not load. HDF5 files on any machine even when experimenting with HDF5 files as large as 800MB did not experience any problems. Clearly, for machines with less memory available and smaller processing capability, HDF5 files are preferred. Defining upper limit processing for both HDF5 and XML files requires additional research and analysis. The upper limit appears to have relationships to processing speed, physical memory, and other constraints. Exploring these limits as a function of different environmental parameters requires recommended future research.

The archiving and processing of large image data requires the use of HDF5, until additional tools and processes are in place that allow for the quick and efficient processing of XML files using computational tools such as MATLAB.

There are many important ethical considerations when sharing data, especially over the Web. Additional considerations to protect the privacy and interests of human participants in data collections require additional guidance when sharing data in a completely public forum where the researcher (and organization) has no control over how the data is used. There will always exist a balance between sharing data for scientific discovery and advancement, and ethical concerns and requirements.

8. References

Barkstrom, B., "Ada 95 Bindings for the NCSA Hierarchical Data Format", proceedings of the 2001 annual ACM SIGAda International Conference on ADA (2001).

Bennett, K., Robertson, J., "The Impact of the Data Archiving Format on the Sharing of Scientific Data for Use in Popular Computational Environments", Proc. SPIE 7687, Orlando Florida (April 2010).

Chapman, S.J., "MATLAB Programming for Engineers; 4th edition; Thomson publishing, Ontario, Canada, 2008.

Dougherty, M., Folk, M., Zadok, E., Bernstein, H., Bernstein, F., Eliceiri, K., Benger, W., Best, C., "Unifying Biological Image Formats with HDF5", communications of the ACM (CACM), 52(10): p. 42-47 (2009).

Gay, L.R. Airasian, P. Educational Research. Prentice Hall, Columbus, Ohio, 2003.

Goetz, A.F.H., Vane, G., Solomon, E., Rock, B.N., "Imaging Spectrometry for Earth Remote Sensing", *Science*, Vol. 228, p. 1147-1153 (1985).

Mather, J., Rogers, A., "HDF5 in MATLAB", Presentation at the HDF5 and HDF-EOS Workshop X, Raytheon System Corporation, Upper Marlboro, MD (November 2007).

MATLAB User's Guide, "Statistical Toolbox, Ttest2", http://www.mathworks.com/help/toolbox/stats/ (April, 2011).

Mauthner, M., Perry, O., "Ethical Issues in Digital Data Archiving and Sharing", eResearch Ethics, http://eresearch-ethics.org (October 2010).

McGrath, R., "XML and Scientific File Formats," Report generated by National Center for Supercomputing Applications, University of Illinois, Urbana-Champaign, in support of work under a Cooperative Agreement with NASA under NASA grant NAG 5-2040 and NAG NCCS-599 (August 2003).

MIT Libraries, "Data Management and Publishing: Ethical and Legal Issues", http://libraries.mit.edu (November 2011).

SAS/STAT User's Guide, "T-Test Procedure", http://support.sas.com/documentation (April, 2011).

Shasharina, S., Li, C., Nanbor, W., Pundaleeka, R., Wade-Stein, D., "Distributed Technologies for Remote Access of HDF Data", proceedings of the 16th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (2007).

Shishedjiev, B., Goranova M., Georgieva, J., "XML-based Language for Specific Scientific Data Description",proceedings of the 2010 Fifth International Conference on Internet and Web Applications and Services (2010).

Sieber, J., "Ethics of Sharing Scientific and Technological Data: A Heuristic for Coping with Complexity & Uncertainty", Data Science Journal, Vol 4, p. 165 (December 2005).

UK Data Archive, “Create and Manage Data - Consent and Ethics: Ethical/Legal/Overview”, http://www.data-archive.ac.uk (2011).

Wang, F., Li, J., Homayounfar, “A Space Efficient XLM DOM Parser”, Data and Knowledge Engineering,Volume 60, Issue 1, p. 185-207 (2007).

MATLAB - A Ubiquitous Tool for the Practical Engineer
Edited by Prof. Clara Ionescu

ISBN 978-953-307-907-3
Hard cover, 564 pages
Publisher InTech
Published online 13, October, 2011
Published in print edition October, 2011

A well-known statement says that the PID controller is the â€œbread and butterâ€ of the control engineer. This is indeed true, from a scientific standpoint. However, nowadays, in the era of computer science, when the paper and pencil have been replaced by the keyboard and the display of computers, one may equally say that MATLAB is the â€œbreadâ€ in the above statement. MATLAB has became a de facto tool for the modern system engineer. This book is written for both engineering students, as well as for practicing engineers. The wide range of applications in which MATLAB is the working framework, shows that it is a powerful, comprehensive and easy-to-use environment for performing technical computations. The book includes various excellent applications in which MATLAB is employed: from pure algebraic computations to data acquisition in real-life experiments, from control strategies to image processing algorithms, from graphical user interface design for educational purposes to Simulink embedded systems.