## A. Summary

- In the area of ***Evaluation and Exploration of Next Generation Systems for Applicability and Performance,*** over the period of 10/1/10 through 12/30/10 the NCSA Innovative Systems Lab team continued to investigate the suitability of the Graphics Processing Unit (GPU) platform for the acceleration of image characterization algorithm implemented in doc2learn application. We have implemented image characterization algorithm used in doc2learn using xpdf C-based library and compared it with the original doc2learn implementation. We have conducted an extensive study to understand and characterize performance of the xpdf-based implementation and to contrast it with the performance of the original doc2learn application in order to point performance bottlenecks and to suggest a strategy for improving doc2learn performance. We have also performed an energy efficiency study of the prior port of doc2lear to the host CPU and GPU platforms and demonstrated that a GPU-based implementation is not power-efficient.

## B. Evaluation and Exploration of Next Generation Systems for Applicability and Performance (Volodymyr Kindratenko, Guochun Shi)

## 1 Preliminaries

Doc2learn implements algorithms for computing *probability density functions* for text, image, and vector graphics objects embedded in PDF files. Specifically, non-parametric probability density function estimation techniques are implemented that require computing a histogram of frequencies of occurrence of all values in a particular object, e.g., image or text. The computed probability density functions (histograms) form the feature vector which can be used for measuring the similarity between pairs of images.

In the last quarter we ported image characterization algorithm used in doc2learn application to C, CUDA C and OpenCL and evaluated its performance on two CPU and two GPU architectures. While the GPU-accelerated version of the algorithm outperformed the original Java implementation for sufficiently large images, one key finding was the identification of the overheads in object extraction from the PDF file due to Java-based PDF handling framework. It became clear that in order to effectively speedup the entire application, performance of the PDF parsing and object extraction framework needs to be improved as well. This quarter we focused on replacing the Java-based framework with a light-weight C-based implementation. Specifically, we used xpdf-3.02 library and modified one of its applications, pdfimage, to compute image histograms identical to the histograms computed by the original doc2learn Java code.

### *1.1 xpdf library*

Xpdf is an open source viewer for PDF files available at http://www.foolabs.com/xpdf. The Xpdf project also includes a PDF text and image extractor, PDF-to-PostScript converter, and various other utilities. The viewer runs under the X Window System on UNIX/Linux and the non-X components (pdfimage, pdftotext, etc.) also run on Win32 systems.

### *1.2 Hardware platform used in this study*

In this study we use Intel Core i7 based system with NVIDIA Fermi GTX 480 GPU. This is the same system used in our prior study as listed in Table 1 of the prior quarter's report.

### *1.3 Energy usage measurements*

To quantify the energy usage by the applications, we used a power monitoring system developer earlier in our lab by Craig Steffen. The power monitor is built in the form of a power distribution unit (PDU), with a C-20 power input port (208 V 16 A) that distributes power to four C-13 power plugs (Figure 1). Complete description of this device can be found in our conference paper[1].

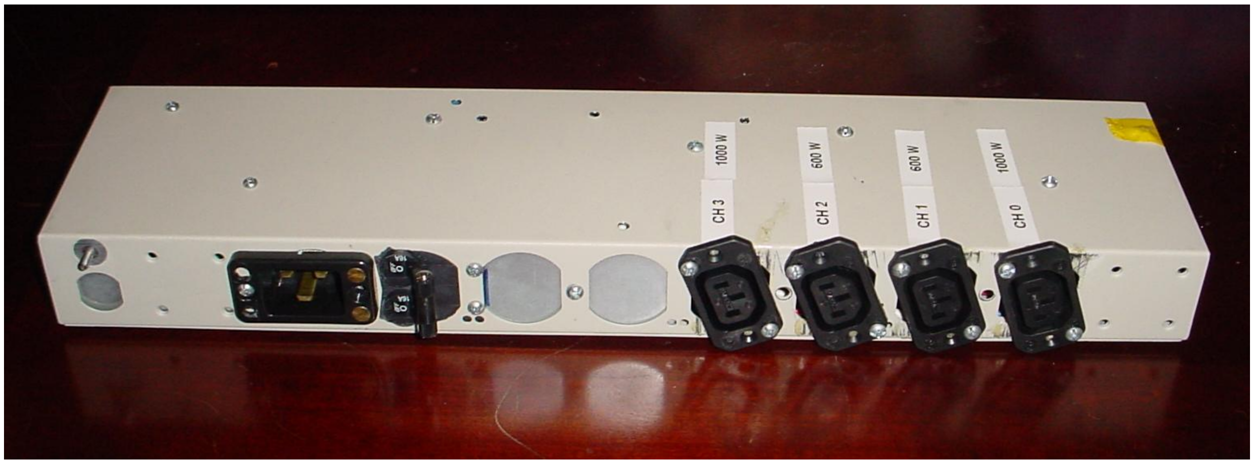


**Figure 1.** Custom-made power monitoring system.

The power monitoring device is connected to the host computer via USB. It outputs measurements as a current in milliamps in ASCII format, which then can be converted into Watts. As part of this project, we have developed data acquisition scripts that capture the output of the power monitor and plot it for further analysis. Figure 2 shows an example plot when monitoring power drawn during the calibration test using a resistor. The load is sequentially moved from one monitored port to another about every 10 seconds to demonstrate that the power measurements remain constant for all monitored ports under load, resulting in about 210 Watt power draw.

[1] J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J. Stone, J. Phillips, *Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters*, In Proc. Work in Progress in Green Computing, 2010.

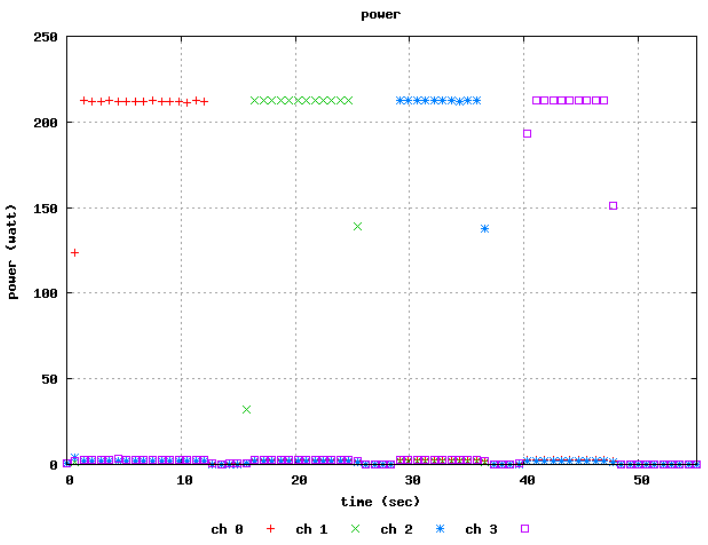


**Figure 2.** Calibration plot. Each of 4 power monitor channels is loaded with a constant load, one channel at a time.

## 2 Technical activities, findings, and results

### *2.1 C-based image extraction and characterization framework*

Our new image extraction and characterization framework is based on xpdf-3.02 library, with the following modifications:

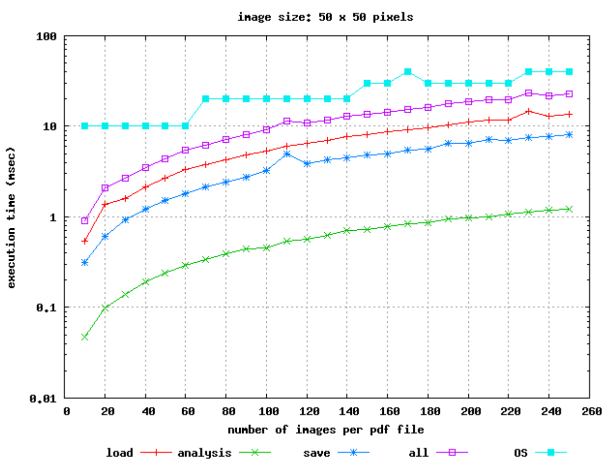
- We replaced `ImageOutputDev::drawImage` method with the code for computing image histogram instead of saving the image to disk. The histogram is computed using the same technique as shown in Figure 8 of the previous quarter's report. Instead of saving a histogram per image as a separate file, we save all the histograms for a given PDF file into a single file with histogram headers uniquely identifying each image. We found that opening and closing multiple files introduces a substantial overhead as this operation makes calls to operating system.
- We added new method, `ImageStream::getLine`, to extract one image row directly into user-supplied memory buffer. This was necessary to eliminate an extra memory copy. Xpdf library still has some inefficiency due to the way image data from a PDF file is loaded via a stream, but eliminating them would require a major rewrite of the stream class as well as some of the methods that use it.

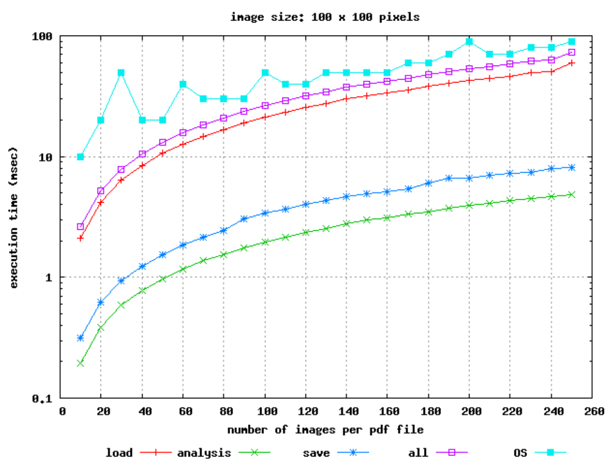
### *2.2 Study to understand the impact of image size*

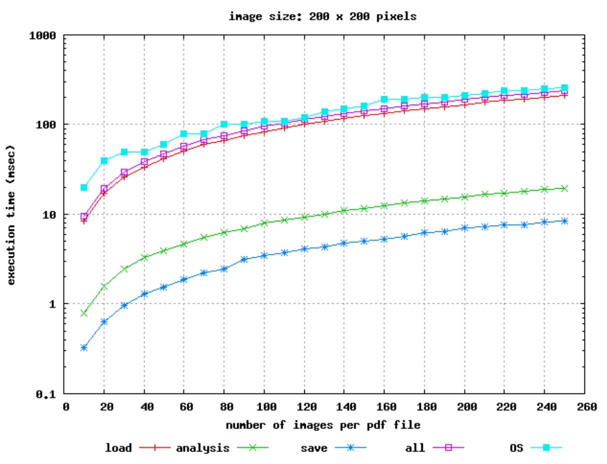
We first investigate performance of our xpdf-based image histogram computation implementation as a function of the image size. In this study we used PDF files with the varying number of images and image sizes generated by Peter Bajcsy. Figure 3 shows execution time as a function of the number of images per PDF file for image sizes 50x50, 100x100, 200x200, 500x500, 1000x1000, and 2000x2000 pixels. We measure performance of different parts of the application: image extraction (load —+—), image analysis (analysis —×—), and histogram storage (save —✱—), as well as the overall image characterization time including extraction, analysis, and storage (all —□—) and the entire application run-time as seen by the operating system (OS —■—). As expected, execution time increases with the number of images per PDF file. However, unexpected observations can be made:

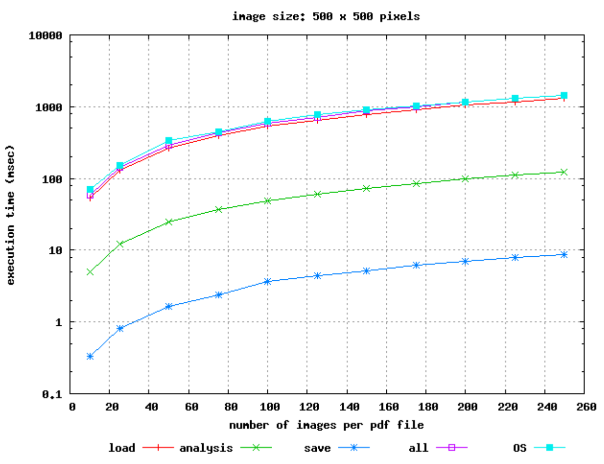
- For small image sizes, histogram extraction time (analysis —×—) is actually less than the time to store the computed histogram in a file (save —✱—). Only for image sizes over ~250x250 pixels the image analysis time becomes larger than the histogram storage time.
- With the exeption of very smal images (i.e., 50x50 pixels), time nessesary to retreive images from the PDF file (load —+—) is about an order of magnitude longer than the time to compute the image histogram (analysis —×—). In other words, file I/O takes about 10 times as long as image processing.
- Operating system (OS) jitter (OS —■—) is very noticeable for small image sizes (short application execution time), but is practically unnoticeable for large image sizes (longer execution time). This requires some additional experimentation to understand the source of such a jitter.

The main conclusion of this study is that accelerating image analysis part does not matter much since image analysis is only responsible for about 10% of the overall application execution time.









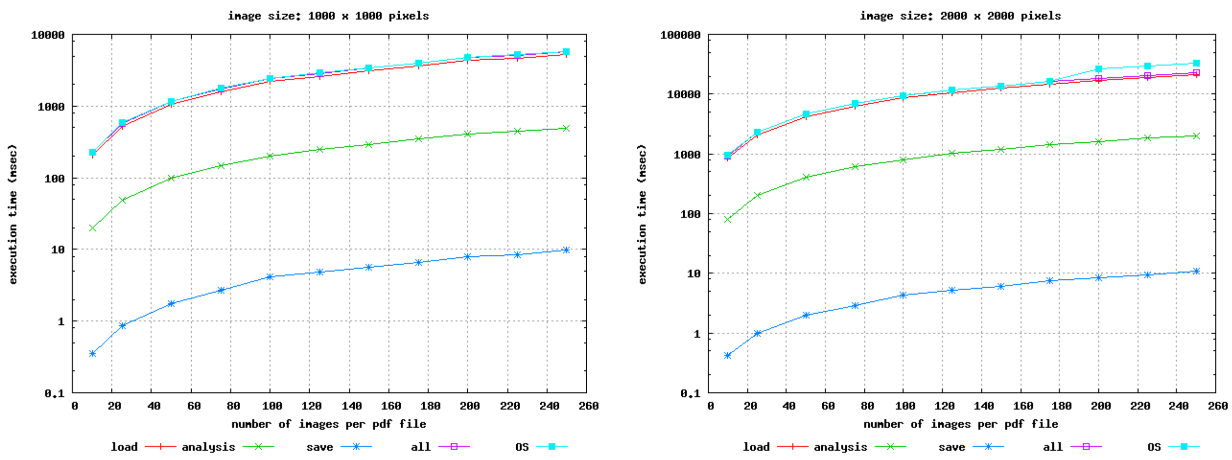


**Figure 3.** Execution time as a function of the number of images per PDF file. Image sizes are 50x50, 100x100, 200x200, 500x500, 1000x1000, and 2000x2000 pixels.

## 2.3 Study to understand the impact of file system

To understand the source of the OS jitter observed in Figure 3 for shorter application execution time, we have conducted an additional study to see the impact of the file system used to store the PDF file. Results plotted in Figure 3 were obtained using local (/tmp) file system. Results plotted in Figure 4 were obtained using network file system (NFS) and RAMDISK file system when executing the application with a set of PDF files containing 200x200 pixels images. Middle-left plot from Figure 3 should be compared to the plots shown in Figure 4.

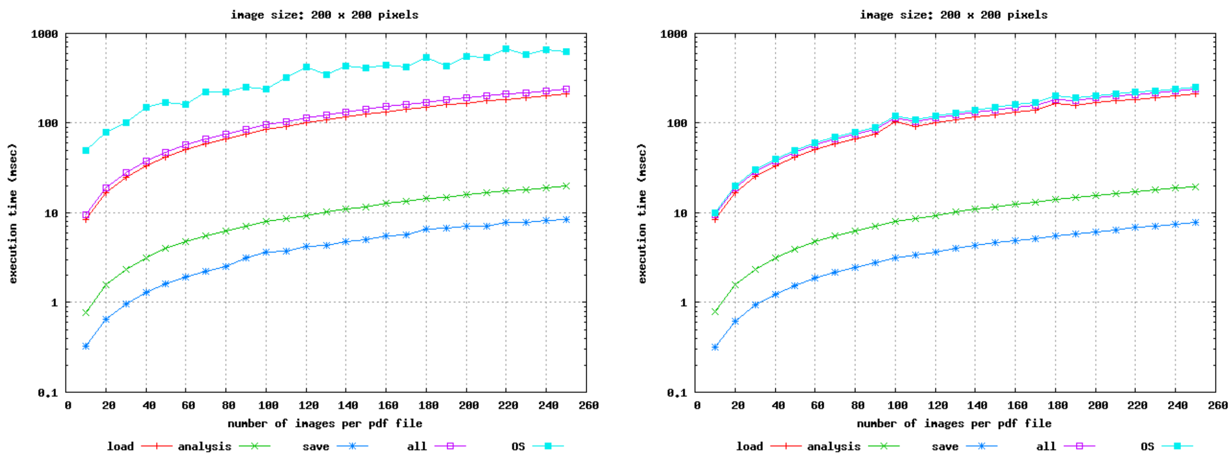


**Figure 4.** Impact of file system. Left: NFS file system; Right: RAMDISK file system.

Based on these measurements, we can make the following observations:

- When the NFS file system is used, a substantial amount of time is spent by the application waiting for the data to arrive from the remote storage. This is expected.

- When the local file system on a spinning disk is used, the shorter the application runtime, the more pronounced the OS jitter is. A spinning disk is a shared resource and other applications and OS itself may use it at the same time our application is accessing a file on the disk.
- When the RAMDISK (memory-based) file system is used to store the PDF file as it is analyzed, the OS jitter is practically absent.

Based on these observations, we can conclude that the RAMDISK file system should be used. A typical usage scenario for doc2learn is to download a set of PDF files for analysis from a remote storage to a local file system and run the analysis on the files from this local storage. If, instead of using the local storage, the files can be put into a RAMDISK, a better application performance should be expected.

## 2.4 Comparison with doc2learn

In Section 3.1, we studied the performance of the xpdf-based image characterization application. Here, we compare this application with the original doc2learn code as well as with our prior C-based implementation of the image characterization algorithm described in the previous quarter's report. Figure 5 shows performance plots for the xpdf-based implementation and the Java and C-based implementations for the same datasets of 200x200 and 1000x1000 pixels per image with varying image count per PDF file. Note that only the image processing time is shown for our previous Java and C-based implementations.

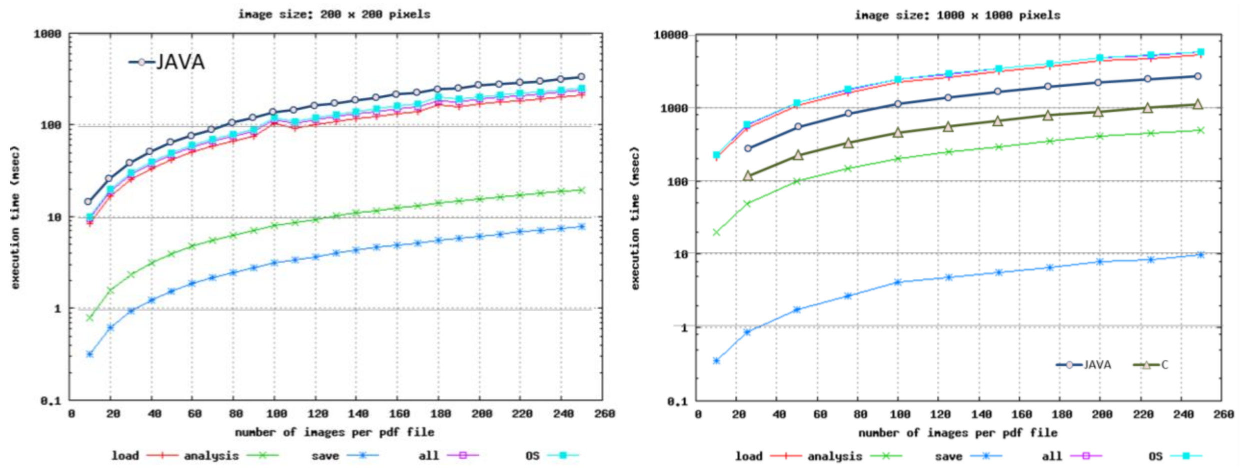


**Figure 5.** Comparing performance of doc2learn with xpdf-based implementation.

From these plots, we can make the following observations:

- Stand-alone xpdf-based implementation is substantially faster than the Java-based doc2learn.
    - For small images, the entire application runs faster than the Java-based image analysis part of doc2learn, not even considering file I/O.
    - For larger images, Java-based image analysis code is still substantially slower

- Reading images from disk takes an order of magnitude more time than to compute histograms, and thus overall application speedup of no more than 10% can be achieved by speeding up the image processing time.

### 2.5 Power efficiency study

The Intel Core i7 platform used in this study is powered through the power monitoring unit described in section 2.3 that allows to measure power consumption of the entire system while it executes an application. Figure 6 includes a plot with the power consumption measurements superimposed for the three versions of doc2learn application: Java-, C-, and GPU-based. The measurements were obtained by first letting the computer idle for 30 seconds, and only then starting up the application while recording power consumption of the host every 1/5th of a second. During the application run with a PDF file with 250 images of 2000x2000 pixels each, the first 132 seconds are spent on parsing the PDF file. Next, image extraction and analysis is executed where image extraction takes 97 seconds and image processing takes 10.3 seconds for Java-, 4.6 seconds for C-, and 3.8 seconds for GPU-based implementations. The idle computer power draw is ~240 watt. When the image extraction and analysis is executed on the host, the power consumption increases to ~260 watt. When the image analysis part is executed on the GPU, the power consumption increases to ~325 watt.

The measurements shown in Figure 6 can be used to characterize power efficiency of the C-based and the GPU-based application as compared to the original Java implementation. We compare wall clock execution time for the original doc2learn run of the image extraction and analysis part of the application, $t$, against the C and GPU-accelerated runs, $t_c$ and $t_g$ respectively, to derive the speedup factors, $s_c=t/t_c$ and $s_g=t/t_g$. Using these speedup factors and the power measurements for the runs, $p$ for Java run, $p_c$ for C run, and $p_g$ for GPU-accelerated run, we can compute the overall improvement in performance-per-watt as $e_c=p/p_c*s_c$ and $e_g=p/p_g*s_g$ This measure shows how many times a GPU-accelerated implementation is more (or less) power-efficient than the CPU-only version of the same application. Summary of the final results is presented in Table I.

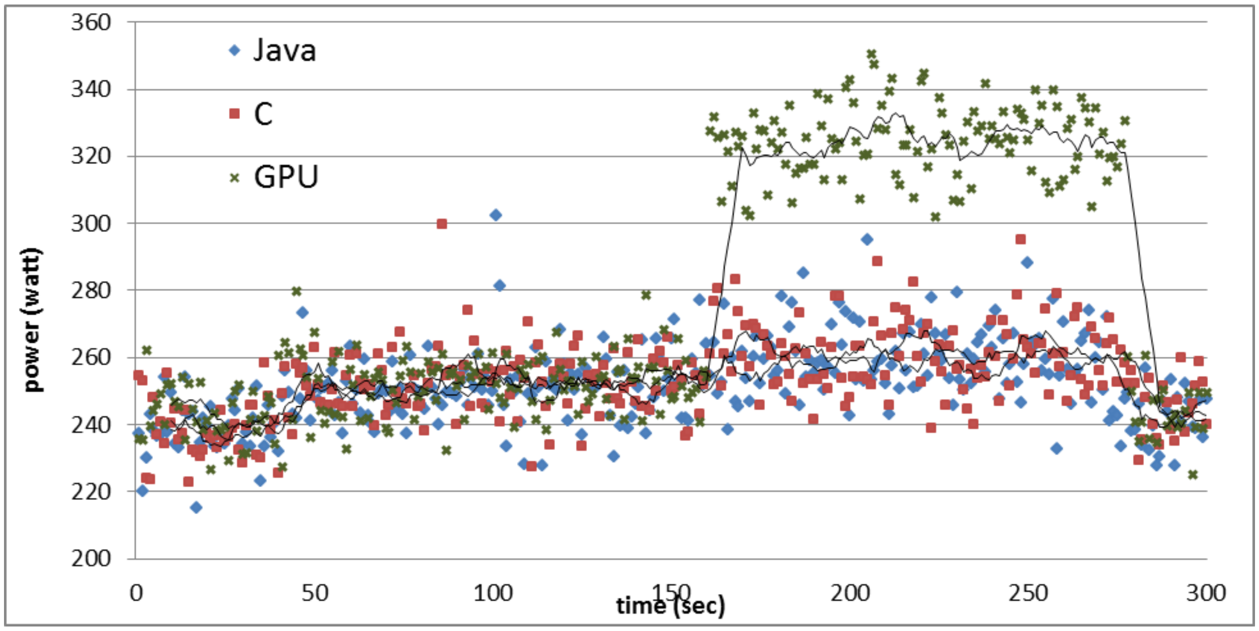


**Figure 6.** Power consumption superimposed for three application runs while processing a PDF file consisting of 250 images of 2000x2000 pixels each.

| | Image analysis only | | Image extraction and analysis | |
|---|---|---|---|---|
| | 1000x1000/250 | 2000x2000/250 | 1000x1000/250 | 2000x2000/250 |
| **t (sec)** | 2.749 | 10.322 | 25.684 | 107.999 |
| **$t_c$ (sec)** | 1.131 | 4.575 | 24.438 | 102.682 |
| **$t_g$ (sec)** | 0.950 | 3.763 | 24.327 | 98.474 |
| **p (watt)** | 260 | 260 | 260 | 260 |
| **$p_c$ (watt)** | 260 | 260 | 260 | 260 |
| **$p_g$ (watt)** | 325 | 325 | 325 | 325 |
| **$s_c=t_c/t$** | 2.43 | 2.26 | 1.05 | 1.05 |
| **$s_g=t_g/t$** | 2.89 | 2.74 | 1.06 | 1.10 |
| **$e_c=p/p_c*s_c$** | 2.43 | 2.26 | 1.05 | 1.05 |
| **$e_g=p/p_g*s_g$** | 2.31 | 2.19 | 0.84 | 0.88 |

**Table I.** Improvement in performance per Watt

As seen from Table I, the C and GPU-based image analysis implementations alone are more power-efficient, by a factor of over 2, than the Java-based implementation. However, this improvement does not translate to the overall application when the image extraction time is taking into account. In fact, the GPU implementation becomes less power efficient than the original Java-based implementation. We have prepared and submitted a conference article describing these experiments and findings[2].

## 3 Conclusions and future work

### 3.1 Suggestions for improving doc2learn performance

Our main recommendation is to re-implement the entire Java-based doc2learn framework in C. Other suggestions are to save all histograms for a given PDF file into just one output file to eliminate multiple fopen/fclose system calls and to use RAMDISK to temporary store PDF files while processing them. We do not recommend using GPUs to accelerate the application since image processing time is only on the order of 1/10th of the overall application execution time. Such a GPU-based implementation will not be power-efficient.

### 3.2 Future work

For the upcoming quarter we will focus our work on two new directions:

- Evaluation of XtremeData database appliance for an efficient data storage and retrieval. The goal of this study is to evaluate the suitability and performance benefits (if any) of a new technology, called database appliance, using NARA-specific datasets as examples. This technology, currently under development by a few vendors, uses hardware acceleration (such as FPGAs) to improve performance of the most commonly used primitive database operations, such as join.
- Study the possibility of accelerating the computation of check sums on large file collections and accelerating data compression using a GPU platform. Computation of check sums for individual files as well as data compression are frequently used operations. In thus study we will evaluate if they can be speedup using a GPU platform and will evaluate if such an acceleration is power- and cost-efficient.

[2] G. Shi, V. Kindratenko, R. Kooper, P. Bajcsy, *GPU Acceleration of an Image Characterization Algorithm for Document Similarity Analysis*, submitted to AICCSA.