

A. Summary

- In the area of *Evaluation and Exploration of Next Generation Systems for Applicability and Performance*, over the period of 01/01/11 through 03/31/11 the NCSA Innovative Systems Lab team investigated the suitability of GPU technology for two important classes of data-intensive applications: *computation of checksums* used for data integrity verification, encryption, and data comparison, and *lossless data compression*. Our findings indicate that while these algorithms can benefit from GPU acceleration to some degree, their practical use on GPUs is limited due to the PCIe bandwidth bottleneck between the host and the GPU. Host memory to CPU bandwidth is substantially higher than host memory to GPU card, thus putting effective limit on performance around 6 GB/s. We also integrated GPU implementation of Scale-Invariant Feature Transformation (SIFT) algorithm with *doc2learn* and *Versus* software and demonstrated its performance for image comparison within these two frameworks.

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

1 Scale-Invariant Feature Transformation

1.1 Summary

Scale-Invariant Feature Transformation (SIFT) is a computer vision algorithm that detects distinctive image features for image matching and recognition. The features are supposed to be invariant to image scaling, rotation, illumination changes, 3D viewpoint change and other affine distortions. The features from an image can be used to compare against those of a sample image and get the similarity estimate of the images.

We have integrated a GPU-based implementation of SIFT algorithm with *doc2learn* pdf file comparison software as a replacement for the probability density function based image comparison algorithm previously used in *doc2learn*. This allows us to compare images embedded in pdf files based on their actual content rather than on the color probability density function.

We also have integrated a GPU-based implementation of SIFT algorithm with *Versus* framework¹ developed by the Image Spatial Data Analysis Group at NCSA.

There are several SIFT implementations available including SIFT++, Oregon state's SIFT implementation³, and SIFT GPU from UNC. In this study, we used the SIFT GPU implementation from UNC² since both feature extraction and feature matching components of the algorithm are implemented in CUDA C providing speedup compared to the reference CPU implementation.

¹ <http://isda.ncsa.illinois.edu/versus/>

² <http://www.cs.unc.edu/~ccwu/siftgpu/>

³ <http://blogs.oregonstate.edu/hess/code/sift/>

1.2 SIFT algorithm performance study

For this comparison, we choose Oregon State's CPU implementation as the reference implementation and UNC implementation as GPU-accelerated implementation and we study how the feature extraction's performance varies as a function of image size. The host CPU we use is 2.67 GHz Intel Xeon and the GPU we use is Nvidia's GTX480, installed in the same CPU host using PCIe interface. In the CPU sift implementation, we use the *siftfeat* program that provides the option to extract features. In the GPU implementation, we modified the sample program *SimpleSift* to do feature extraction only. The run time is measured using the *time* utility invoked from the command line. Because the GPU always have a startup overhead time, which can be invoked by calling *cudaFree(0)* at the beginning of the program, this startup time is subtracted in the final GPU run time as it is a one-time cost. The CPU/GPU run time with different image sizes is shown in Figure 1. For small images, CPU runs faster than GPU. However, as the image size increases, the CPU run time increases substantially while the GPU run time stays relatively stable. With image size 2592x1936, the GPU implementation runs almost 4x faster than CPU implementation.

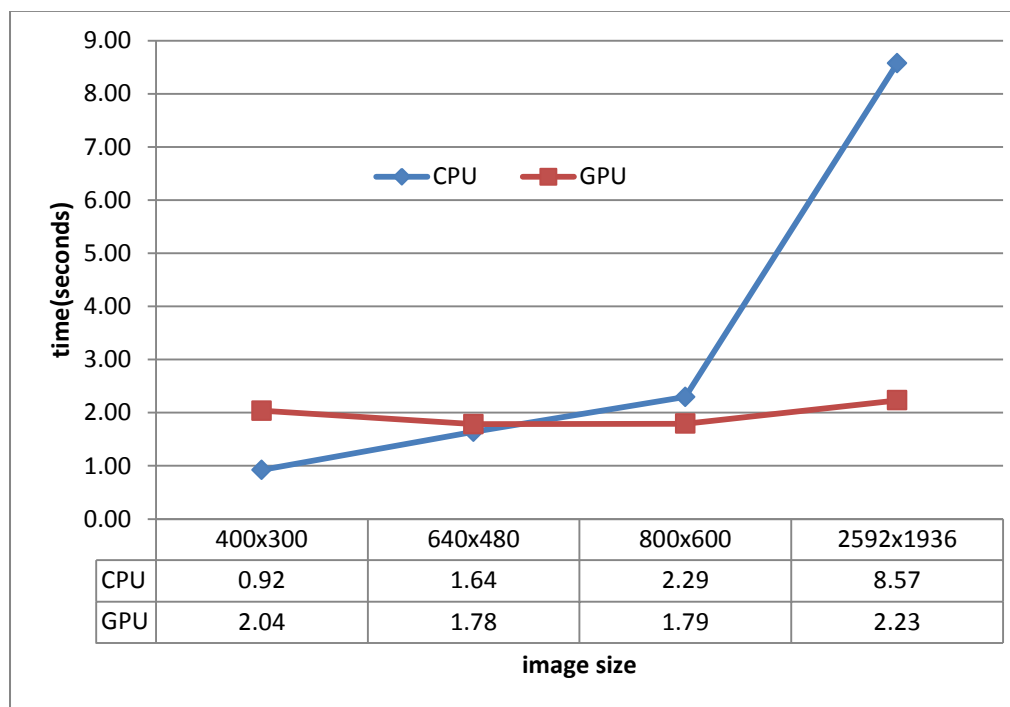


Figure 1. Performance comparison between CPU and GPU implementation of SIFT algorithm.

1.3 SIFT integration with Versus

Versus is a framework currently under development in the Image Spatial Data Analysis Group at NCSA to facilitate the comparison of digital objects. The framework exposes a JAVA-based API for extending its functionality by adding new comparison methods. Figure 2 gives an overview of the API.

We implemented four new classes on top of the four interfaces, Adapter, Extractor, Measure, and Descriptor, in order to integrate SIFT GPU program with *Versus*.

- Class `SiftGpuAdaper` implements `Adapter` interface and the method `getBytes()` calls through Java Native Interface (JNI) `SiftGpu` library to load the image file data into memory.
- Class `SiftGpuExtractor` implement the interface `Extractor` and the method `extract()` calls through JNI `SiftGpu` library to compute all the features in the image and return the feature data back.
- All feature data is stored in class `SiftGpuFeature`, which implements the interface `Descriptor`.
- Class `SiftGpuMeasure` implements the interface `Measure` and the method `compare()` calls through JNI `SiftGpu` library to invoke the comparison performed on the GPU to compute matching features.

API Overview

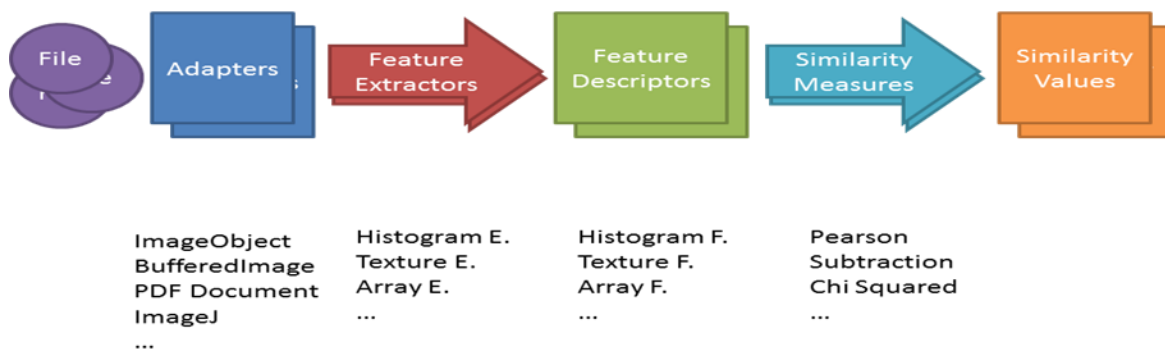


Figure 2. *Versus* API overview.

Based on the matching features and the total number of features in both images, one can compute a similarity number for the two images. The `SiftGpu` library code is modified accordingly to provide the interface to the JNI calls. Other auxiliary methods in the four classes such as `getName()` are also implemented to make the `SiftComparison` method work seamlessly within the *Versus* framework. Figure3 demonstrates the configuration process for selecting `SiftGpu` library methods within *Versus* framework.

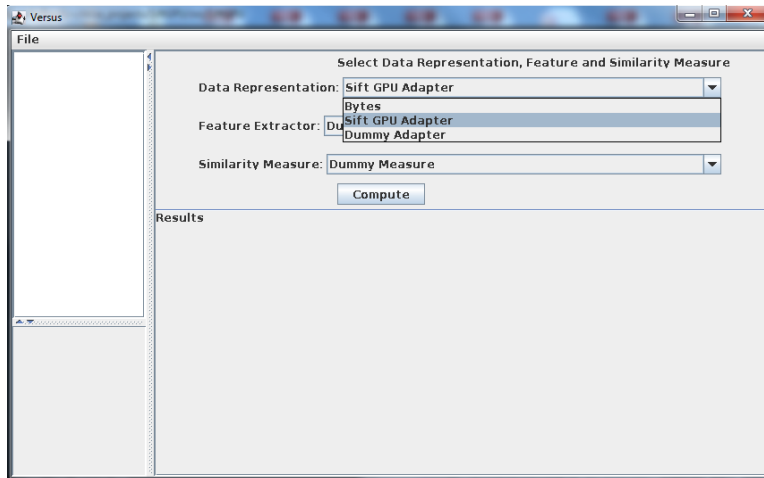


Figure 3a. Selection of SiftGpuAdapter.

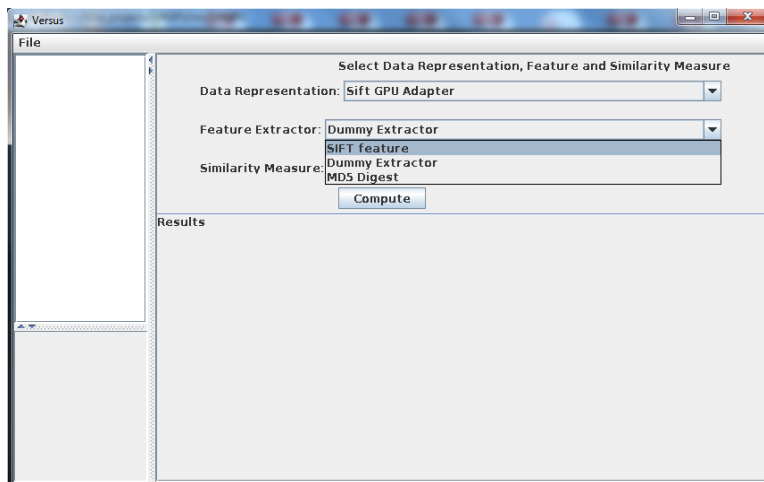


Figure 3b. Selection of SiftGpuExtractor.

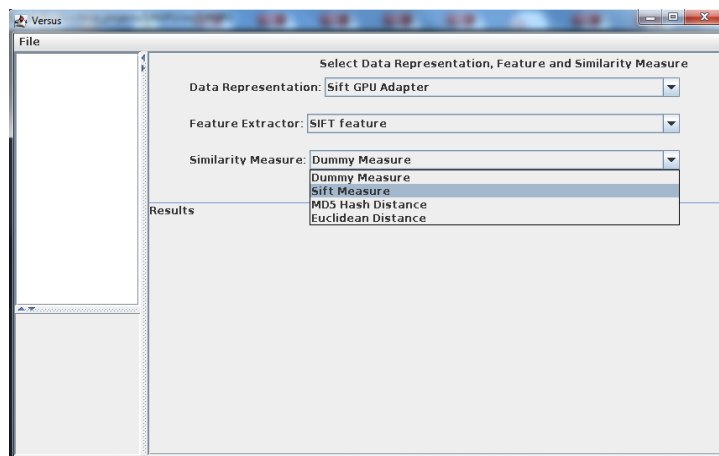
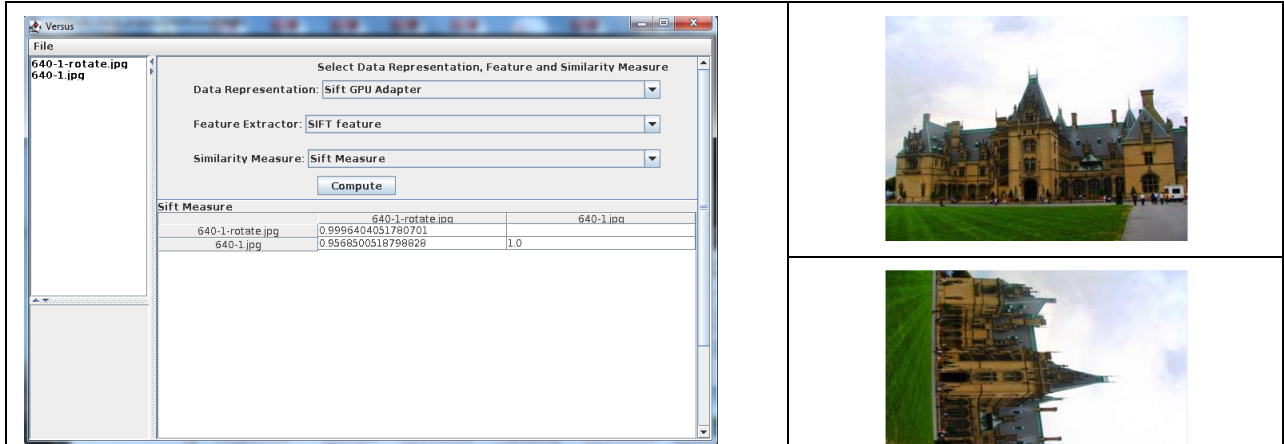


Figure 3c. Selection of SiftGpuFeature.

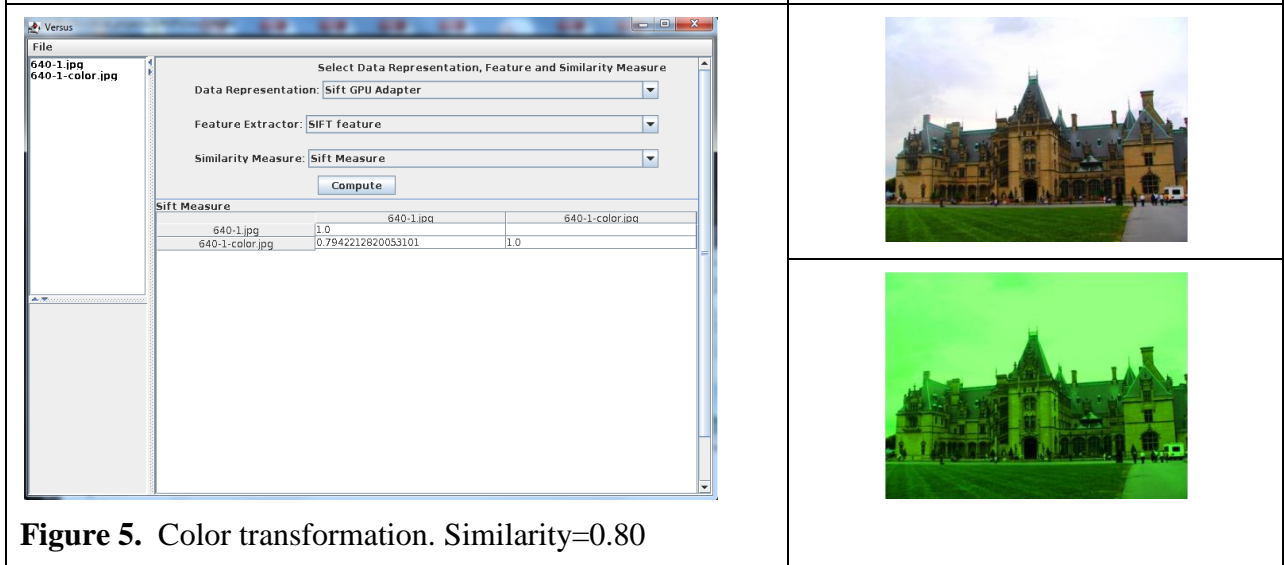
1.4 Examples

The examples below demonstrate the use of SIFT integrated with *Versus* using few image pairs. In the current implementation, a number ranging from 0 to 1 is used to indicate the degree of similarity between pairs of images.



Sift Measure		
	640-1-rotate.jpg	640-1.jpg
640-1-rotate.jpg	0.9996404051797791	
640-1.jpg	0.9568500518798828	1.0

Figure 4. Rotation. Similarity=0.96



Sift Measure		
	640-1.jpg	640-1-color.jpg
640-1.jpg	1.0	
640-1-color.jpg	0.7942212820053101	1.0

Figure 5. Color transformation. Similarity=0.80

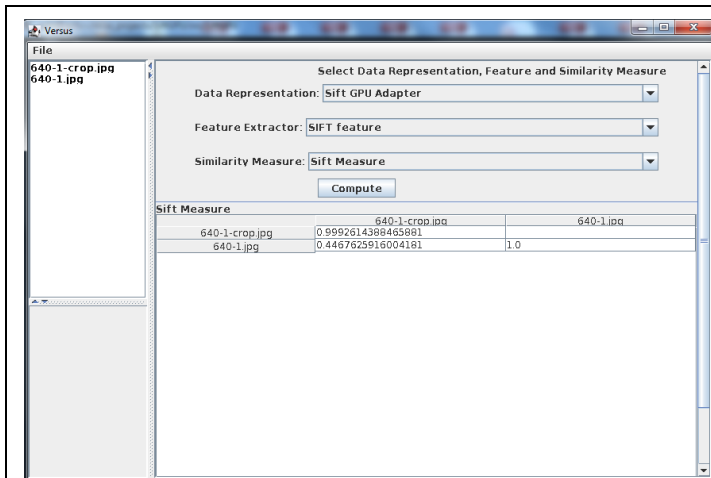


Figure 6. Image subsection comparison. Similarity=0.45

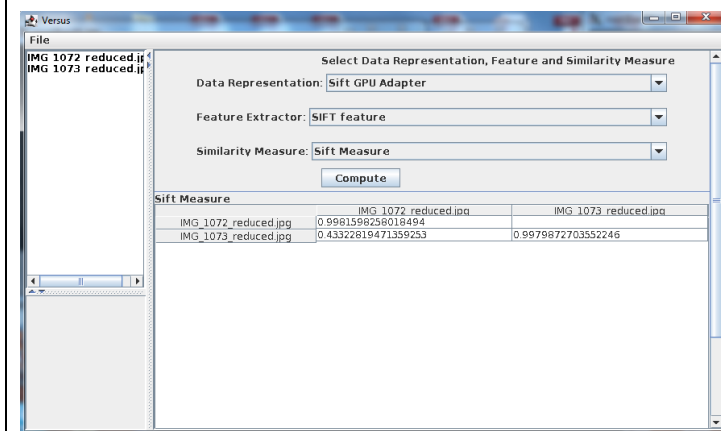


Figure 7. Different light exposure. Similarity=0.43

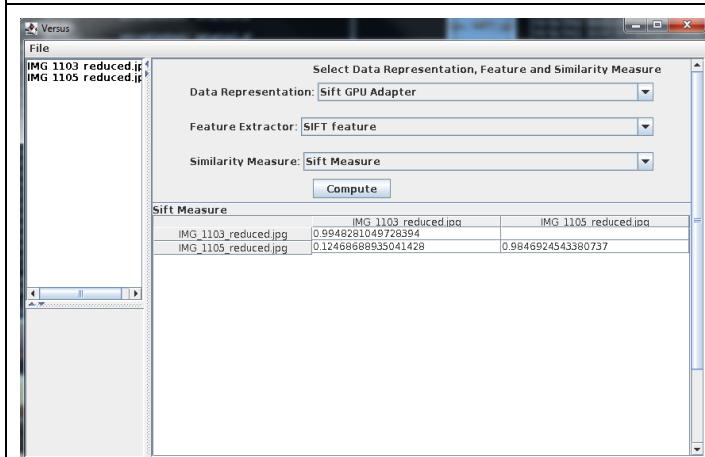
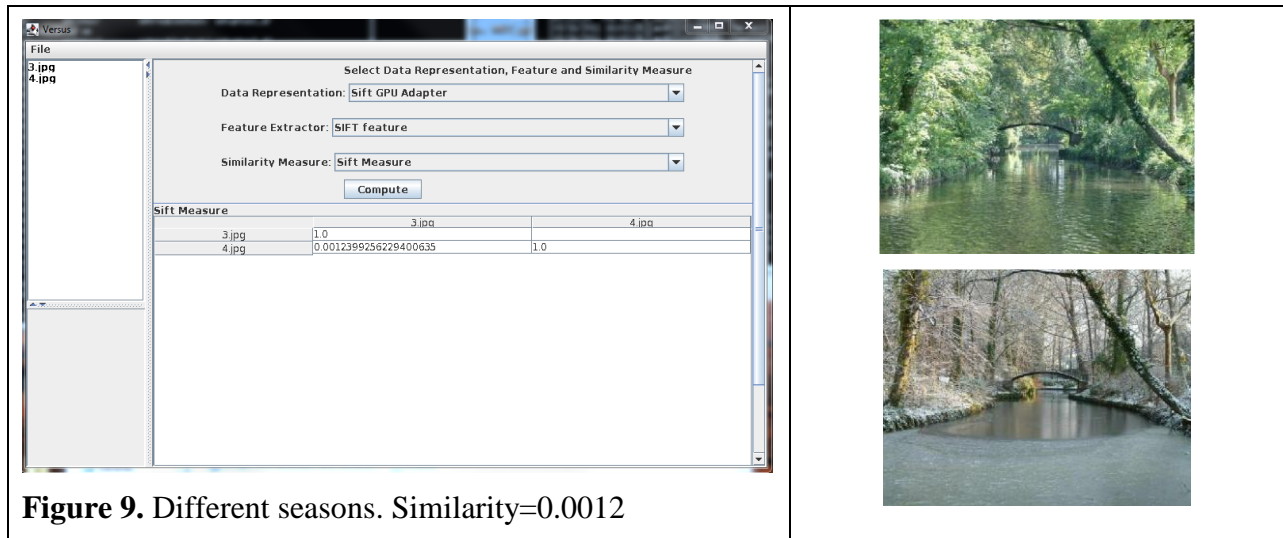


Figure 8. Different angle. Similarity=0.12





There is current one limitation when using SiftGpu library within *Versus* framework. The SiftGpu code has a default limit of 4096 features used for comparison. While there is an API to change the default behavior, it appears to be not fully functional as of time of writing this report. Manually changing the MACRO definition in the code caused the “unspecified launch error” on the GPU, which is a likely indication of oversubscription of GPU resources, e.g., the shared memory in this case. Further testing and code improvement is needed.

1.5 SIFT integration with doc2learn

The integration of SiftGpu with doc2learn follows the same approach we used when integrating GPU-based doc2learn image histogram computation code. The image data, width and height are passed to SiftGpu library through JNI calls in the histogram computation function in the Java code and the Sift features are computed by the SiftGpu library and saved in files. Normally one image has hundreds or thousands of features where each feature is represented by 4 floating values, representing location (x, y), scale and orientation, and 128 integers describing the feature. These features can later on be used for image comparison.

2 Lossless data compression

Variety of lossless data compression techniques exist, including

- Dictionary-based encoders (Lempel-Ziv family of algorithms) that replace input strings with pointers to a dictionary data structure updated by the encoder.
- Variable-length entropy encoders, such as Huffman and arithmetic coding, that assign prefix codes to each input symbol based on its statistical frequency of occurrence.
- Run-length coders that store repeated values as a single instance and a count.

Best lossless data compression techniques work by using probabilistic models in which predictions are coupled to an arithmetic coding algorithm.

To date, no serious prior work has been done on general-purpose lossless data compression on GPUs. The closest is 2009 students work from Stanford University³. But there is some good prior work for floating-point data compression⁴. The floating point data compression idea is based on the suppression of leading zeros in the residuals between the input values and their predicted values. This technique however is not suitable for general-purpose use and its performance is limited by the PCIe bus bandwidth (~6 GB/sec).

We have conducted analysis of data compression techniques with regards to their suitability for GPU implementation. In the process of doing so, we re-implemented parts of the Lempel-Ziv (LZ) compression method which is at the core of the many dictionary-based encoders. This implementation was necessary to understand the data flow through the algorithm.

Our conclusion is that existing general-purpose lossless data compressor algorithms are not amenable for an efficient GPU implementation due to:

- Serial nature of the algorithms
 - Parallelization via data partitioning is possible, but not sufficient to utilize massive parallelization of GPUs
 - Control instructions abundant nature of the algorithms
 - Irregular data access patterns
- PCIe bandwidth
 - Currently peak at 8 GB/s for PCIe gen. 2 x16
 - Compared to 10-17 GB/s for DDR3 memory modules

We have not yet found a general-purpose lossless data compression algorithm amenable for an efficient GPU implementation.

3 Check sums

National Institute of Standards and Technology (NIST) announced a public competition on Nov. 2, 2007 to develop a new cryptographic hash algorithm, in response to the recent development of cryptanalysis. The winner will be named “SHA-3” and will subject to a Federal Information Processing Standard (FIPS). In Dec 2008, 51 first round candidates were announced and the number narrowed down to 14 in July 2009 in the second round. In December 2010 the final five candidates were selected by NIST to enter the third (and final) round of SHA-3 competition and the winner is expected to be announced in 2012.

The candidates are reviewed based on security, cost and algorithmic and implementation characteristics. The GPU implementation of these algorithms is of great interests as GPGPU gains momentum in high performance computing. The first round candidate md6 has been implemented in GPU and used in fast file matching⁵. Bos, etc. has evaluated the performance of all of the second round candidates (BLAKE, Grøstl, JH, Keccak, and Skein) on the Nvidia GPU

³ L. Wu, M. Storus, D. Cross, CUDA Compression Project, http://ppl.stanford.edu/cs315a/pub/Main/CS315a/CUDA_Compression_Final_Report.pdf

⁴ M. O’Neil, M. Burtscher, Floating-Point Data Compression at 75 Gb/s on a GPU, GPGPU-4, Mar 05-05 2011.

⁵ Deephan Mohan, John Cavazos, Faster File Matching Using GPGPUs, SAAHPC2010

platform and implemented most of them⁶. Since cryptographic hash function usually contains only bits operations, GPUs can achieve very high performance given enough parallelism. However, since the original message data is in host memory and has to be copied to GPU through PCIe, the best performance cannot exceed the PCIe bandwidth (~6 GB/s). Indeed, in Bos's GPU implementations, the best possible performance is achieved with algorithms "BLAKE-32" and "BMW-256" algorithms, both being able to compute 36.8 Giga bits per second or 4.6 GB/s, approaching the PCIe limit.

4 Future work

We have planned to evaluate performance of XtremeData database appliance⁷ for data analysis on the datasets relevant to NARA. However, during this reporting period we were unable to obtain access to the hardware. XtremeData finally installed the data analytics appliance, called dbx, on March 30 and we plan to conduct an in-depth evaluation of this technology in the upcoming month.

⁶ Joppe W. Bos and Deian Stefan. Performance analysis of the SHA-3 candidates on exotic multi-core architectures. CHES 2010

⁷ <http://www.xtremedata.com/-support/downloads/category/14-dbx>