Performance-Analysis-Based Acceleration of Image Quality Assessment

Thien Phan*, Sohum Sohoni†, Damon M. Chandler*,[1]
*Laboratory of Computational Perception and Image Quality
†Laboratory of Computer Architecture Education, Simulation, and Research
School of Electrical and Computer Engineering
Oklahoma State University
Stillwater, OK 74078 USA
(thien.phan,sohum.sohoni,damon.chandler)@okstate.edu

Eric C. Larson
DUB Group
Department of Electrical Engineering
University of Washington
Seattle, WA 98195 USA
eclarson@uw.edu

***Abstract*—Two stages are commonly employed in modern algorithms of image/video quality assessment (QA): (1) a local frequency-based decomposition, and (2) block-based statistical comparisons between the frequency coefficients of the reference and distorted images. This paper presents a performance analysis of and techniques for accelerating these stages. We specifically analyze and accelerate one representative QA algorithm recently developed by the authors (Larson and Chandler, 2010). We identify the bottlenecks from the abovementioned stages, and we present methods of acceleration using integral images, inline expansion, a GPGPU implementation, and other code modifications. We show how a combination of these approaches can yield a speedup of 47x.**

***Keywords*-Image quality, video quality, integral image, GPGPU, acceleration, code optimization.**

I. Introduction

Image and video quality assessment (QA) have begun to play important roles in the design, operation, and validation of numerous systems. However, as QA moves from the research community into more mainstream applications, the bottlenecks of current algorithms are starting to prevent widespread adoption. Indeed, modern image QA algorithms such as VIF [1], MS-SSIM [2], and MAD [3] are quite effective at QA, but they require a relatively large run-time—on the order of seconds. As these algorithms are adapted to process frames of video (e.g., MOVIE [4], ST-MAD [5]), run-time issues become of greater importance.

The bulk of computation and run-time of QA algorithms occurs in two stages: (1) local frequency-based decompositions of the reference and distorted images; and (2) statistical comparisons between the local frequency coefficients, typically implemented in a block-based fashion. For example, in MS-SSIM an image is decomposed into different scales and local image statistics are compared in a moving window. In VIF, wavelet subband covariances are computed and can be compared via a sliding window approach. While acceleration of local frequency decompositions has received considerable attention as a general image processing tool (e.g., [6], [7]), the acceleration and parallelization of local statistical comparisons has largely been ignored.

In this paper, we present a performance analysis of and methods for accelerating a representative QA algorithm recently developed by the authors, MAD [3]. MAD employs a log-Gabor decomposition and a comparison of local statistical differences between blocks of log-Gabor coefficients of the reference and distorted images; it is thus an appropriate representative algorithm for the stages performed in QA.

We identify the bottlenecks in MAD, and we present four methods for accelerating the algorithm: (1) Using integral images for the statistical computations; (2) using procedure expansion and strength reduction; (3) using a general-purpose-GPU (GPGPU) implementation of the log-Gabor decomposition; and (4) precomputation and caching of the log-Gabor filters. As we will show, a combination of these approaches can result in a nearly 47x speed increase. While our implementation is specific to MAD, our methodology is straightforward to apply to a variety of QA algorithms; the analysis and results presented here can also provide insight into how other related algorithms might be accelerated.

II. MAD: Description and Performance Analysis

A. Description of the MAD Algorithm

The MAD algorithm consists of two assessment stages: (1) a detection-based stage, which estimates quality based on the extent to which the *distortions* are visible; and (2) an appearance-based stage, which estimates quality based on the extent to which the *image* is recognizable.

Due to space limitations, we refer readers to [3] for further details of the detection-based stage. Our main focus in this paper is on the appearance-based stage, which employs a computational neural model using a log-Gabor filterbank with both even-symmetric and odd-symmetric filters applied using the FFT [8]. The even and odd filter outputs are combined to yield magnitude-only subband values. The variance, skewness, and kurtosis computed for each 16×16 block (with 75% overlap between blocks) of each subband of the original image are compared to corresponding values

[1]This material is based upon work supported by, or in part by, the National Science Foundation Award #1054612, and by the U.S. Army Research Laboratory (USARL) and the U.S. Army Research Office (USARO) under contract/grant number W911NF-10-1-0015.

Table I
SYSTEM 1 WAS USED FOR THE MAIN PERFORMANCE ANALYSIS; SYSTEMS 2 AND 3 WERE USED FOR THE ADDITIONAL TIMING RESULTS SHOWN IN TABLE II).

	CPU and RAM	OS and Matlab
System 1 (Desktop)	Intel Core 2 Quad Q9400 2.66 GHz 8 GB 1333-MHz DDR3	Win. 7, 64-bit Matlab 2011a
System 2 (Laptop)	Intel Core 2 Duo T6400 2 GHz 4 GB 1333-MHz DDR3	Win. 7, 32-bit Matlab 2009a
System 3 (Server)	Two Intel Xeon 5050 3 GHz 5 GB 667-MHz DDR2	Win. Server 2003 Matlab 2009a

computed for the distorted image. The differences between these statistics are then collapsed via a 2-norm to yield a scalar output corresponding to appearance-based differences.

B. Performance Analysis

The original version of MAD was implemented in Matlab, with C++ MEX files used for computation of the block-based statistics. To identify bottlenecks, we performed a timing analysis using the profiler in Matlab. The MEX files were compiled using Microsoft Visual C++ 2008 using Matlab's default optimization flags `/O2 /Oy-`.

The performance analysis was executed on a Dell Inspiron 580 desktop, and for verification on two additional systems. Table I lists the specifications of these systems (additional details can be found here: http://vision.okstate.edu/MAD/Acceleration/).

We ran MAD on 180, 512×512-pixel images from the CSIQ database [9]. Figure 1 shows the average results from this analysis for System 1 (similar distributions of time were observed on all three systems); the average total execution time per image is 55.85 seconds. As shown in Figure 1(a), over 98% of this time is spent in the appearance-based stage. This stage consists of three sub-stages: a log-Gabor decomposition, computation of block-based statistics of the log-Gabor subbands to generate statistical difference maps, and combining/collapsing the maps into a final scalar output. As shown in Figure 1(b), computation of the statistical difference maps alone consumes over 93% of MAD's total execution time. Thus, performance improvement should focus on this part of the algorithm.

III. INTEGRAL IMAGES AND INLINE EXPANSION

A. Integral Images

To accelerate the computation of the statistical difference maps, we employ and build upon a technique called *integral images* originally developed in the context of computer graphics [10]. The integral image, which is also known as the *summed area table*, is an algorithm for quickly computing the sum of values within any block of an image.

Let I denote an image (or subband) for which one needs to compute block-based sums. The integral image M has the

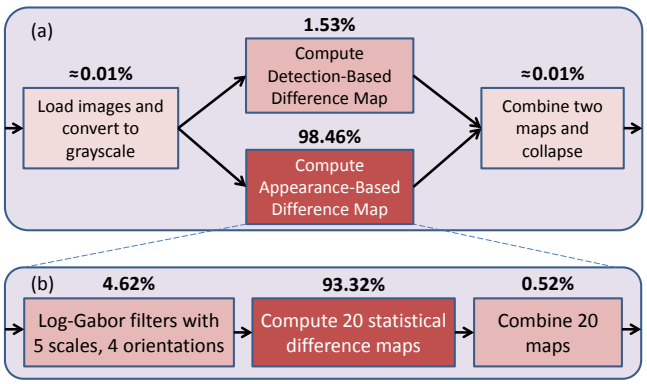

Figure 1. (a) Profiling analysis showing average execution time in each stage for System 1; the average total execution time per image is 55.85 seconds. (b) Breakdown of the bottleneck, the appearance-based stage.

same dimensions as I, but with each pixel value at position (x, y) given by:

$$M(x,y) = \sum_{y'=1}^{y} \sum_{x'=1}^{x} I(x',y'). \quad (1)$$

Given M as defined above, the sum s of all pixel values located in the rectangle (x_1,y_1), (x_2,y_2) is given by

$$s = M(x_2,y_2) + M(x_1,y_1) - M(x_1,y_2) - M(x_2,y_1). \quad (2)$$

A further reduction in computation can be achieved by capitalizing on the fact that MAD does not need to compute the statistics of *every* block. MAD calculates the statistical maps using blocks of size 16×16 pixels, with 12 pixels of overlap between neighboring blocks. Accordingly, we do not need the entire integral image, just subsets of it required to support the computation of the statistics within these particular blocks. In this case, the integral image M has dimensions 1/4 of the size of I, and is given as follows:

$$M(x,y) = \sum_{y' \le 4y} \sum_{x' \le 4x} I(x',y'). \quad (3)$$

Unfortunately, integral images were designed to compute a block's sum, whereas MAD requires standard deviation, skewness, and kurtosis. Specifically, let $\mathbf{b}$ denote a $N_1 \times N_2$ block of I. The standard deviation, skewness, and kurtosis of $\mathbf{b}$ are given by

$$\sigma_{\mathbf{b}} = \sqrt{\frac{1}{N_1 N_2} \sum_i \left(\mathbf{b}_i - \bar{\mathbf{b}}\right)^2}, \quad (4)$$

$$\varsigma_{\mathbf{b}} = \frac{1}{N_1 N_2 \sigma_{\mathbf{b}}^3} \sum_i \left(\mathbf{b}_i - \bar{\mathbf{b}}\right)^3, \quad (5)$$

$$\kappa_{\mathbf{b}} = \frac{1}{N_1 N_2 \sigma_{\mathbf{b}}^4} \sum_i \left(\mathbf{b}_i - \bar{\mathbf{b}}\right)^4, \quad (6)$$

where b_i and $\bar{\mathbf{b}}$ denote the i^{th} pixel and mean of $\mathbf{b}$.

We developed an extension of the integral image algorithm to accelerate the computation of Equations (4)-(6). This modification requires computing integral images for the image and powers of the image up to power 4.

Table II
TIMING RESULTS OF VARIOUS MODIFICATIONS. THE FIRST AND SECOND COLUMNS FOR EACH SYSTEM SHOW RUN-TIME IN SECONDS AND SPEEDUP, RESPECTIVELY. II = INTEGRAL IMAGE; IE = INLINE EXPANSION OF `pow`.

Mod.	System 1		System 2		System 3		Average	
None	55.8	1.0x	56.42	1.0x	62.23	1.0x	58.15	1.0x
II	6.77	8.2x	8.84	6.4x	10.98	5.7x	8.86	6.6x
IE	3.77	14.8x	7.90	7.1x	9.45	6.6x	7.04	8.3x
II + IE	3.21	17.4x	5.46	10.3x	7.38	8.4x	5.35	10.9x

Specifically, let M_1, M_2, M_3, and M_4 denote the integral image computed for I, I^2, I^3, and I^4, respectively. Let s_1, s_2, s_3, and s_4 denote sums of the values (over the same coordinates as block $\mathbf{b}$) in I, I^2, I^3, and I^4, respectively.

To compute the standard deviation, we manipulate Equation (4) as follows:

$$\begin{aligned}
\sigma_{\mathbf{b}} &= \sqrt{\frac{1}{N_1 N_2}\sum_i \left(\mathbf{b}_i^2 - 2\mathbf{b}_i\bar{\mathbf{b}} + \bar{\mathbf{b}}^2\right)}, \\
&= \sqrt{\frac{1}{N_1 N_2}\left(\sum_i \mathbf{b}_i^2 - 2\sum_i \mathbf{b}_i\bar{\mathbf{b}} + N_1 N_2\bar{\mathbf{b}}^2\right)}, \\
&= \sqrt{\frac{1}{N_1 N_2}\left(s_2 - 2s_1\frac{s_1}{N_1 N_2} + N_1 N_2\left(\frac{s_1}{N_1 N_2}\right)^2\right)}, \\
&= \sqrt{\frac{1}{N_1 N_2}\left(s_2 - \frac{s_1^2}{N_1 N_2}\right)}.
\end{aligned}$$

Similar manipulation of Equations (5) and (6) result in the following formulas for $\varsigma_{\mathbf{b}}$ and $\kappa_{\mathbf{b}}$.

$$\begin{aligned}
\varsigma_{\mathbf{b}} &= \frac{1}{N_1 N_2 \sigma_{\mathbf{b}}^3}\left(s_3 - 3\frac{s_1 s_2}{N_1 N_2} + 2\frac{s_1^3}{(N_1 N_2)^2}\right), \\
\kappa_{\mathbf{b}} &= \frac{1}{N_1 N_2 \sigma_{\mathbf{b}}^4}\left(s_4 - 4\frac{s_1 s_3}{N_1 N_2} + 6\frac{s_2 s_1^2}{(N_1 N_2)^2} - 3\frac{s_1^4}{(N_1 N_2)^3}\right),
\end{aligned}$$

The results of applying this modification will be presented shortly (see Section III-C).

B. Procedure Inlining and Strength Reduction

In the original implementation of MAD, Equations (4), (5), and (6) are implemented using the `pow` function from the Standard C++ Library. Another avenue for accelerating the computation of the statistics is to replace the function call with a direct multiplication.

We therefore replaced the calls to `pow` with an inline expansion involving a direct multiplication (e.g., `pow(x,3)` was replaced with `x*x*x`). This modification not only saves the overhead related to the procedure call, but also offers strength reduction.

C. Results

The timing results of these modifications are shown in Table II. Both approaches clearly provide a significant acceleration. However, it is important to note that the integral image (II) and inline expansion (IE) offer different forms of acceleration. The II aims to minimize the number of power computations, whereas the IE aims to make each power computation faster. Thus, even though the results shown in Table II indicate that the IE offers more acceleration, the II can offer more acceleration when the block size or number of blocks increases.[1] It is therefore prudent to use both approaches. Indeed, as shown in Table II, when both modifications are employed, MAD is accelerated by nearly 11x on average (17x for System 1).

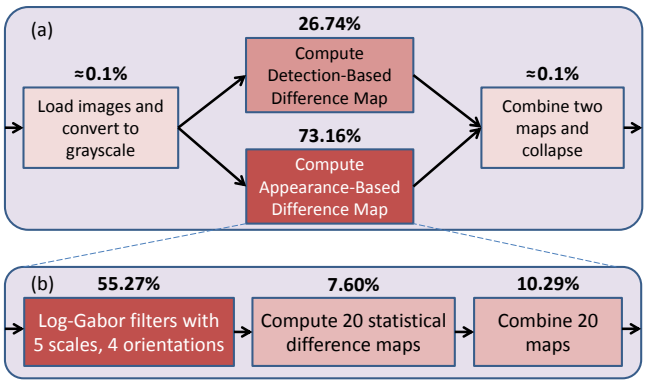

Figure 2. Updated version of Figure 1; the average total run-time per image is 3.21 secs (System 1). The log-Gabor filtering is now the bottleneck.

Figure 2 shows an updated version of Figure 1, indicating the distribution of time required by each portion of MAD when using II+IE. These latter results indicate that the bottleneck is no longer in the computation of the statistics, but is rather in the computation of the log-Gabor decomposition. In the following section, we describe a GPGPU-based acceleration of the log-Gabor decomposition.

IV. GPGPU AND OTHER CODE OPTIMIZATIONS

MAD's log-Gabor decomposition is performed for both the original image and the distorted image using twenty log-Gabor filters which span five scales and four orientations; the filtering results in 40 total subbands (20 subbands each for the original and distorted image). This process is well suited to a parallel implementation because each subband can be computed separately from all other subbands.

In MAD, the log-Gabor decomposition is implemented entirely in Matlab. To parallelize this decomposition, we employed Matlab's GPGPU (CUDA) facilities. MATLAB treats the functions used in the log-Gabor code as GPU functions. Because the GPGPU implementation requires specific hardware, only System 1 was tested. This system used an NVIDIA GeForce GTX 560 Ti with 1 GB of RAM with Version 8.17.12.8026 of NVIDIA's driver.

Table III shows the results of the GPGPU implementation (second row) along with two additional code modifications.

[1]For example, when using 16×16 blocks with 14 pixels (rather than 12 pixels) of overlap between neighboring blocks, the time required for the II approach increases by approximately 10%, while the time required for the IE approach increases by approximately 190%.

Table III
TIMING RESULTS OF GPGPU AND OTHER CODE MODIFICATIONS (SYSTEM 1 ONLY). CO = CODE OPTIMIZATION; IE-DT = INLINE EXPANSION OF POW IN MAD'S DETECTION-BASED STAGE

Modification	Run-time in secs.	Speedup vs. Orig.	Speedup vs. II+IE
II + IE (repeated from Table II)	3.21	17.4x	1.0x
II + IE + GPGPU	2.04	27.3x	1.6x
II + IE + GPGPU + CO	1.75	31.9x	1.8x
II + IE + GPGPU + CO + IE-DT	1.19	47.0x	2.7x

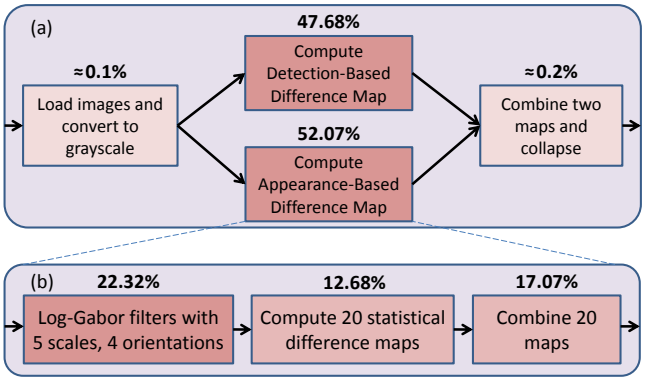

Figure 3. Updated version of Figure 2 using all acceleration techniques; the average total execution time per image is 1.19 seconds. The log-Gabor decomposition has reduced to 22% of the total time (previously 55%).

The third row shows the results when the log-Gabor code is further optimized by precomputing the filters and caching them in RAM. The last row shows the results of applying inline expansion of the pow function to all parts of MAD (including the preprocessing and detection-based stages). When all of these acceleration techniques are employed, MAD's run-time is accelerated by 47x.

When combined with II+IE, the GPGPU implementation provides a lower-than-expected 1.6x speedup over II+IE alone. The GPGPU implementation required copying the images to the GPU, and then copying the resulting log-Gabor subbands from GPU back to the CPU for the statistical computations. We suspect that the overhead involved with these memory transfers reduced the overall performance gain. This finding is consistent with our previous study on CUDA [11], which revealed that the memory bandwidth between the system and the GPU can create a bottleneck that reduces the potential gains.

Figure 3 shows an updated version of Figure 2, indicating the new distribution of times. Notice that the log-Gabor decomposition has been reduced from its previous value of approximately 55% to the new value of 22%. The appearance-based and detection-based stages of MAD now consume a more balanced share of the execution time.

V. CONCLUSION

This paper presented techniques for accelerating the two most computationally expensive stages employed in many QA algorithms: local frequency-based decomposition, and local statistical comparisons between the frequency coefficients of the reference and distorted images. We specifically analyzed and accelerated one representative QA algorithm, MAD [3]. The results of our performance analysis showed that the bottlenecks stem from these stages, and we presented four methods of acceleration. Although this study focused on one specific algorithm, our methodology and acceleration techniques are applicable to a variety of QA algorithms.

We are currently in the process of extending this work through the use of performance counters and other profiling techniques. Such an analysis will yield further insight into the hardware resource usage of the different stages. Another avenue for acceleration that we are currently investigating is to change the algorithm in an effort to reduce its computational complexity (e.g., reducing the number of the filters used in the decomposition, or simplifying the statistical computations used to compare the coefficients). These latter approaches are likely to change the output of the algorithm; however, it is possible to still achieve a reasonable tradeoff between predictive performance and overall run-time.

REFERENCES

[1] H. R. Sheikh and A. C. Bovik, "Image information and visual quality," *IEEE Transactions on Image Processing*, vol. 15, no. 2, pp. 430–444, 2006.

[2] Z Wang, E P Simoncelli, and A C Bovik, "Multiscale structural similarity for image quality assessment," in *Proc 37th Asilomar Conf on Signals, Systems and Computers*, Pacific Grove, CA, Nov 9-12 2003, vol. 2, pp. 1398–1402, IEEE Computer Society.

[3] E. C. Larson and D. M. Chandler, "Most apparent distortion: full-reference image quality assessment and the role of strategy," *Journal of Electronic Imaging*, vol. 19, no. 1, 2010.

[4] K. Seshadrinathan and A.C. Bovik, "Motion tuned spatio-temporal quality assessment of natural videos," *Image Processing, IEEE Transactions on*, vol. 19, no. 2, pp. 335 –350, Feb 2010.

[5] C. Vu and D. M. Chandler, "Main subject detection via adaptive feature refinement," *Journal of Electronic Imaging*, vol. 20, no. 1, March 2011.

[6] Nikolaus Voßand Bärbel Mertsching, "Design and implementation of an accelerated gabor filter bank using parallel hardware," in *Proceedings of the 11th International Conference on Field-Programmable Logic and Applications*, London, UK, 2001, FPL '01, pp. 451–460, Springer-Verlag.

[7] I.T. Young, L.J. van Vliet, and M. van Ginkel, "Recursive gabor filtering," *Signal Processing, IEEE Transactions on*, vol. 50, no. 11, pp. 2798 – 2805, nov 2002.

[8] P. D. Kovesi, "MATLAB and Octave functions for computer vision and image processing," Centre for Exploration Targeting, School of Earth and Environment, The University of Western Australia, Available from: <http://www.csse.uwa.edu.au/∼pk/research/matlabfns/>.

[9] E. C. Larson and D. M. Chandler, "Categorical image quality (csiq) database," Online, http://vision.okstate.edu/csiq/.

[10] Franklin C. Crow, "Summed-area tables for texture mapping," *SIGGRAPH Comput. Graph.*, vol. 18, pp. 207–212, Jan 1984.

[11] B. Gordon, S. Sohoni, and D. Chandler, "Data handling inefficiencies between cuda, 3d rendering, and system memory," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, dec. 2010, pp. 1 –10.