GPU-Based Single Image Dehazing: Implementation and Optimization (Postprint)

Authors: Zhang Jin, Zhou Xiangquan, Shu Man, Wang Yulan, Wei Youhua, Liu Bingli

Date: 2018-05-20T00:00:00+00:00

Abstract

Dehazing algorithms based on dark channel prior have achieved promising dehazing results, but their computational time is excessively long, failing to meet real-time dehazing requirements. The dehazing algorithm was preliminarily implemented in parallel using GPU, and the components requiring optimization were identified. During the optimization process, data were stored in high-speed memory to enable rapid data access, while a novel algorithmic implementation was designed to reduce computational complexity, ultimately improving the speedup ratio. The optimized accelerated algorithm requires only 21 ms to process a 768×1024 image, satisfying the real-time dehazing requirements.

Full Text

Preamble

Title: Implementation and Optimization of Single Image Haze Removal Based on GPU

Authors: Zhang Jin, Zhou Xiangquan, Shu Man, Wang Yulan†, Wei Youhua, Liu Binli
(Geomathematics Key Laboratory of Sichuan Province, Chengdu University of Technology, Chengdu 610059, China)

Abstract: The defogging algorithm based on dark channel prior has achieved excellent results, but its computational time is too long to meet real-time defogging requirements. This paper uses GPU to initially implement the defogging algorithm in parallel and identifies the portions requiring optimization. During the optimization process, we store data in high-speed memory to enable rapid data access and design a new algorithm implementation to reduce computational complexity, ultimately improving the acceleration ratio. The optimized

accelerated algorithm requires only 21 ms to process a 768×1024 image, meeting the requirements for real-time defogging.

Keywords: image defogging; GPU; parallel optimization; real-time fog removal

Classification: TP303
DOI: 10.3969/j.issn.1001-3695.2017.07.0889

0 Introduction

Atmospheric scattering by suspended particles such as fog and haze reduces scene visibility, causing attenuation in contrast and color saturation of captured images. In computer vision and image processing applications—including object detection, outdoor surveillance, and remote sensing image processing—high-quality images that faithfully restore real scenes are essential. Moreover, real-time applications such as navigation and autonomous driving demand not only effective defogging but also rapid implementation.

Among defogging techniques, the method based on dark channel prior proposed by He et al. has gained prominence due to its superior performance and broad applicability. Initially, this method employed soft matting to refine the transmission map, which incurred high computational complexity. He et al. later replaced soft matting with guided filtering, which offers linear computational complexity while maintaining comparable defogging quality, making it one of the most effective defogging algorithms available.

However, even with guided filtering, the algorithm remains computationally intensive. Processing a 600×400 image using soft matting requires approximately 10–20 seconds, while guided filtering reduces this time but still falls short of real-time requirements. For higher-resolution images, the computation speed remains prohibitively slow. Since NVIDIA introduced the CUDA interface in 2007, researchers have leveraged GPU for rapid defogging. For instance, Lvy et al. achieved 0.083 seconds for a 600×400 image using GPU, and Xue et al. reported processing times around 0.036 seconds. While these efforts significantly reduced computation time, acceleration for high-definition images remains insufficient.

This paper delves deeper into optimization strategies for algorithm acceleration, focusing on two key aspects: efficient utilization of high-speed memory and reduction of algorithmic complexity. These improvements substantially increase the acceleration ratio, enabling near real-time processing of high-definition images.

1 Algorithm Overview

The widely used haze formation model is derived from atmospheric scattering:

$$I(x) = J(x)t(x) + A(1 - t(x))$$

where $I(x)$ denotes the hazy input image, $J(x)$ represents the haze-free image to be recovered, A is the atmospheric light, and $t(x)$ is the transmission map. The goal of defogging is to recover $J(x)$ from $I(x)$. The dark channel prior states that in haze-free images, pixel values in the dark channel are extremely low and approach zero.

Based on this prior, the defogging algorithm proceeds as follows [Figure 1: see original paper]:

a) Compute dark channel. For each pixel in the original image, the dark channel is computed within a local patch $\Omega(x)$:

$$\text{dark}(x) = \min_{y \in \Omega(x)} \left(\min_{c \in \{r,g,b\}} J^c(y) \right) \approx 0$$

b) Estimate atmospheric light. Select the top 0.1% brightest pixels in the dark channel, retrieve their corresponding values in the original image, and compute their average as the atmospheric light estimate.

c) Compute coarse transmission. From the dark channel prior, we derive:

$$t(x) = 1 - \omega \frac{\text{dark}(x)}{A}$$

Since the atmosphere naturally contains some particles, completely removing haze may produce unnatural results. We introduce a parameter ω (fixed at 0.95 following prior work) to retain a small amount of haze.

d) Optimize transmission using guided filtering. Directly using coarse transmission produces block artifacts. Guided filtering optimizes the transmission map. Let I be the guidance image (the grayscale version of the original image, as in He' s MATLAB code), p the coarse transmission, and q the refined transmission. Within local windows, we compute linear coefficients a and b:

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p}_k - a_k \mu_k$$

The final transmission is obtained by averaging:

$$q_i = \bar{a}_i I_i + \bar{b}_i$$

e) Recover image. To avoid noise amplification when transmission is low, we set a lower bound $t_0 = 0.1$:

$$J(x) = \frac{I(x) - A}{\max(t(x), t_0)} + A$$

This paper focuses on acceleration and optimization; detailed algorithmic procedures and formulas can be found in references [6,7]. The defogging results are shown in [Figure 2: see original paper], where (a) and (c) are hazy input images, and (b) and (d) are the restored results. Pixel values match those computed by the reference MATLAB code.

---

2 Parallel Acceleration and Optimization

GPU parallel acceleration using the CUDA platform requires understanding its hardware architecture. CUDA devices consist of scalable streaming multiprocessors (SMs). When a kernel is launched, SMs allocate resources in thread blocks, executing multiple threads concurrently within each block. Each thread handles computation for one or more pixels, following CUDA' s SIMT (Single Instruction, Multiple Thread) execution model.

Our design maps one thread to each pixel, organizing thread blocks and grids in 2D to correspond with image pixels [Figure 3: see original paper]. To identify performance bottlenecks, we profiled the CPU implementation [Figure 4: see original paper], revealing that dark channel computation and transmission optimization dominate execution time. We therefore focus optimization efforts on these two steps.

2.1 Dark Channel Computation

The computational complexity for dark channel computation is approximately $m \times n \times r \times r \times 3$, where r is the patch radius and $m \times n$ is image size. Beyond high computational cost, implementation challenges include boundary handling and redundant data access.

Consider a patch radius of 3. For edge pixels like the top-left corner [Figure 5: see original paper], five neighboring pixels fall outside the image boundary, requiring extensive conditional checks that significantly slow computation. Moreover, adjacent pixels redundantly load overlapping data (e.g., pixels (1,4) and (3,4) both load the three yellow blocks in [Figure 5: see original paper]).

To address this, we preload data into high-bandwidth memory. While registers offer the fastest access, they are private to each thread and limited in capacity.

Shared memory provides a better solution, enabling data reuse among threads within a block while maintaining fast access.

Data Loading Strategy: For a thread block (white region in [FIGURE:6(a)]), each thread loads its own pixel plus neighboring "halo" pixels (gray region) into shared memory. [FIGURE:6(b)-(d)] illustrate how threads load overlapping regions. Although adjacent thread blocks still exhibit redundant reads of boundary data [Figure 7: see original paper], the overall reduction in global memory access yields significant speedup.

Performance Comparison: [Figure 8: see original paper] shows that using shared memory achieves an average 7× speedup over global memory for dark channel computation.

2.2 Transmission Optimization

The transmission optimization involves equations (5)-(8). Direct implementation requires only two kernel launches after fusion, but demands careful handling of boundaries, shared memory size, and coalesced access. In contrast, guided filtering requires over twenty kernel launches with repeated box_filter invocations. We therefore compared both approaches across different window sizes [Figure 9: see original paper].

Results show that box_filter becomes increasingly advantageous as window size grows. The box_filter parallelization comprises row-wise and column-wise accumulation, with remaining operations being simple matrix computations. Thus, optimizing accumulation efficiency is crucial.

Parallel Reduction Optimization: Standard parallel accumulation on 8 elements requires 17 additions [Figure 10: see original paper]. For N elements, this needs $N\log_2(N) - (N-1)$ operations—inefficient and scaling poorly. Our optimized approach [Figure 11: see original paper] reduces this to $2N - \log_2(N) - 2$ operations, demonstrating linear growth and substantial computational savings [Figure 12: see original paper].

Key Optimization Principles: 1. **Exploit high-speed memory:** Reuse data across threads via shared memory; store thread-private data in registers when possible. 2. **Reduce algorithmic complexity:** Different implementations of the same algorithm offer varying parallel efficiency. Prioritize approaches that maintain parallelism while lowering complexity.

Future work will focus on coalesced global memory access patterns when loading data to shared memory and minimizing kernel launch overhead during transmission optimization.

---

3 Experimental Results

The experimental environment is summarized in .

Table 1: Experimental Environment

Hardware Environment
AMD FX(tm)-6300 Six-Core Processor 3.50GHz 8.00GB RAM GeForce GTX970 GPU, 4GB VRAM

Software Environment
Microsoft Windows 10 (64-bit Edition) Microsoft Visual Studio 2013 CUDA Toolkit 7.5

Processing times for various image sizes are shown in .

Table 2: Computation Time for Different Image Sizes

Image Size	CPU Version	GPU Initial Version	GPU Optimized Version
300×400			
400×600			
600×800			
768×1024			21 ms

The optimized version achieves approximately 10× speedup over the initial GPU implementation. For high-resolution 768×1024 images, processing takes less than 21 ms—nearly 50 frames per second—satisfying real-time requirements.

4 Conclusion

Numerous studies have accelerated defogging algorithms, and our approach indeed outperforms previous work on 600×400 images. However, direct comparison is difficult due to varying experimental environments. The key contribution of this paper lies in proposing optimization strategies:

1. **Maximize high-speed memory utilization** to reduce global memory access. Data shared among threads within a block should reside in shared memory; small, thread-private data can use registers.
2. **Reduce algorithmic complexity** to minimize computation. Different implementations of the same algorithm exist; prioritize those that preserve parallelism while lowering complexity.

Future directions include optimizing coalesced memory access patterns for shared memory loading and reducing kernel launch overhead during transmission optimization.

References

[1] Narasimhan S G, Nayar S K. Contrast restoration of weather degraded images [J]. IEEE Trans on Pattern Analysis Machine Intelligence, 2003, 25 (6): 713-724.

[2] Liu Qi, Gao Xinbo, He Lihuo, et al. Haze removal for a single visible remote sensing image [J]. Signal Processing, 2017, 137: 33-43.

[3] Hung C L, Ma Zhaohui, Lin Chunyuan, et al. Image haze removal of optimized contrast enhancement based on GPU [J]. Frontier Computing, 2016, 375: 53-63.

[4] 郭璠, 蔡自兴, 谢斌, 等. 图像去雾技术研究综述与展望 [J]. 计算机应用, 2010, 30 (9): 2417-2421.

[5] He Kaiming, Sun Jian, Tang X. Single image haze removal using dark channel prior [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2011, 33 (12): 2341-2353.

[6] He Kaiming, Sun Jian, Tang Xiaoou. Guided image filtering [J]. IEEE Trans on Pattern Analysis and Machine Intelligence, 2013, 35 (6): 1397-1409.

[7] Lvy Xingyong, Chen Wenbin, Shen I F. Real-time dehazing for image and video [C]// Proc of the 18th IEEE Conference on Pacific Computer Graphics and Applications. 2010.

[8] Xue Y G, Ren Ju, Su Huayou, et al. Parallel implementation and optimization of haze removal using dark channel prior based on CUDA [J]. High Performance Computing, 2013, 207: 99-109.

[9] 李佳童, 章毓晋. 图像去雾算法的改进和主客观性能评价 [J]. 光学精密工程, 2017, 25 (3): 735-741.

[10] 魏颖慧, 张彦娥, 梅树立, 等. 基于暗通道先验和区间插值小波变换的图像去雾霾方法 [J]. 农业工程学报, 2017, 33 (S1): 281-287.

[11] 宋颖超, 罗海波, 惠斌, 等. 尺度自适应暗通道先验去雾方法 [J]. 红外与激光工程, 2016, 45 (9): 286-297.

[12] 陈书贞, 任占广, 练秋生. 基于改进暗通道和导向滤波的单幅图像去雾算法 [J]. 自动化学报, 2016, 42 (3): 455-465.

[13] Tarel J P, Hautiere N. Fast visibility restoration from a single color or gray level image [C]// Proc of the 12th IEEE International Conference on Computer Vision. 2009.

[14] Nicholas Wilt. 2014. The CUDA Handbook: A Comprehensive Guide to GPU Programming [M]. Beijing: China Machine Press.

Note: Figure translations are in progress. See original paper for figures.