Project Proposal:
Parallel Bilateral Filtering Implementation and Optimization

Group members: Zhaowei Zhang, Eric Zhu

Project Website: https://pentene.github.io/parallel-bilateral-filtering

Summary

In this project, we will implement optimized parallel versions of the bilateral filtering algorithm on CPU and GPU platforms, specifically using OpenMP and CUDA, respectively. We aim to benchmark these implementations rigorously, analyzing their performance characteristics and scalability across diverse optimization strategies and hardware configurations.

Background

Bilateral filtering is a widely-used image processing technique known for its effectiveness in smoothing images while preserving edges. This edge-preserving smoothing is accomplished by calculating the filtered value for each pixel as a weighted average of neighboring pixels, considering both spatial proximity and intensity similarity. Specifically, for a pixel p, the bilateral filter computes the filtered pixel intensity $I_p^{'}$ as:

$$I'(p) = \frac{1}{W_p} \sum_{q \in \Omega} I(p) k_r(\|I(q) - I(p)\|) k_s(\|p - q\|)$$

and the normalization term W, defined as

$$W_p = \sum_{q \in \Omega} k_r(\|I(p) - I(q)\|) k_s(\|p - q\|)$$

where

- I and I' are the input and output images
- k_r and k_s are the range and space kernels each with its own definition
- σ_r is the intensity range smoothed out and σ_s is the smoothing factor

Here is a basic Bilateral Filtering Algorithm (Pseudocode):

```
for each pixel p in image do
    normalization = 0;
    filteredPixel = 0;
    for each neighbor q within window do
        spatialWeight = k_r(||I(q) − I(p)||);
        intensityWeight = k_s(||p − q||);
        weight = spatialWeight * intensityWeight;
        filteredPixel += weight * I(p);
        normalization += weight;
    end
    outputImage[p] = filteredPixel / normalization;
end
```

Algorithm 1: Sequential Bilateral Filter

The bilateral filter's per-pixel computation involves intensive arithmetic operations and memory access, making it computationally expensive, particularly with large neighborhood windows or high-resolution images. Each pixel's computation involves accessing neighboring pixels within a defined radius repeatedly. Nonetheless, it exhibits good potential for parallelization since each pixel's final output computation is independent of other pixels, even though there are neighborhood dependencies. Exploiting parallelism through CPU multithreading (OpenMP) and GPU massive threading (CUDA) can reduce execution time and enhance efficiency.

Goals and Deliverables

Here are our planned goals:

- Achieve at least a 4x speedup with CPU parallelization using OpenMP compared to the sequential baseline.
- Achieve at least a 10x speedup with GPU parallelization using CUDA compared to the sequential baseline.
- Perform a thorough comparative analysis of performance and scalability between CPU and GPU implementations using image datasets such as High Resolution Image Quality (HRIQ) and The USC-SIPI Image Database.
- Validate correctness by comparing results with the OpenCV bilateral filtering implementation.

Here are our aspirational goals:

- Achieve a 10x speedup on CPU with robust scalability across varying image sizes.
- Achieve a speedup exceeding 50x on GPU through advanced optimizations such as FFT-based fast convolution techniques inspired by Sylvain Paris and Frédo Durand's work ("A Fast Approximation of the Bilateral Filter using a Signal Processing Approach").

Here are our fallback goals:

- Ensure fully functional parallel implementations with measured performance gains, even if the desired speedup targets are not fully met.
- Complete a rigorous analysis clearly identifying bottlenecks and documenting performance insights.

Methodology

The project will be structured as follows:

1. Implement a straightforward sequential bilateral filter. This will be our baseline performance metric for benchmarking

2. Parallelize the algorithm using OpenMP. Investigate optimizations such as loop scheduling, cache-aware blocking, and SIMD vectorization. Then measure performance gains and identify bottlenecks on CPU.

3. Develop a GPU-based implementation that uses basic CUDA kernels. Profile and measure performance improvements over sequential and parallel CPU implementations. On this basis, explore advanced GPU optimizations, including shared-memory utilization, memory coalescing, occupancy tuning, and optimal thread/block configurations.

4. Perform a comprehensive comparison of sequential, parallel CPU and GPU implementations. Evaluate scalability, speedup, efficiency, and memory performance across various image sizes and configurations.

Challenges

Parallelizing the bilateral filtering algorithm is challenging due to several critical factors:

- Workload dependencies: Pixel computations are not fully independent, as each pixel calculation involves accessing neighboring pixels within a defined spatial radius. However, each pixel's output calculation remains independent of other pixel outputs, allowing parallel computation.

- Memory access characteristics: Although bilateral filtering exhibits good spatial locality, it is still important to use cache-aware techniques and shared memory optimizations to exploit this locality.

- Communication-to-Computation ratio: While arithmetic computations involved are relatively simple (exponential functions, intensity comparisons, and distance calculations), each calculation requires multiple memory accesses. Without effective optimization, the ratio of memory accesses to computations can be high. However, optimized implementations using caching or shared memory can significantly lower this ratio.

- Divergent execution: Intensity-dependent calculations within the bilateral filter can lead to divergent execution patterns, particularly on GPUs.

- System constraints: The algorithm's performance heavily relies on GPU shared memory or CPU cache capacities. If the neighborhood window exceeds these capacities, frequent global memory accesses become unavoidable, degrading overall performance.

Through addressing these challenges, we aim to gain deep insights into optimizing memory access patterns, synchronization overhead, and computational efficiency in parallel systems.

Resources

Implementation and benchmarking will be performed primarily on GHC lab machines and Pittsburgh Supercomputing Center (PSC) resources. Initial sequential implementations will be written from scratch, with OpenCV's bilateral filter function used for correctness verification. PSC machines, which offer more robust CPU and GPU capabilities, will be particularly useful for large-scale tests and performance analysis.

Here is a list of references:

- Tomasi, C., & Manduchi, R. (1998, January). Bilateral filtering for gray and color images. In Sixth international conference on computer vision (IEEE Cat. No. 98CH36271) (pp. 839-846). IEEE.
- Paris, S., & Durand, F. (2006, May). A fast approximation of the bilateral filter using a signal processing approach. In European conference on computer vision (pp. 568-580). Berlin, Heidelberg: Springer Berlin Heidelberg.
- Paris, S., Kornprobst, P., Tumblin, J., & Durand, F. (2009). Bilateral filtering: Theory and applications. Foundations and Trends® in Computer Graphics and Vision, 4(1), 1-73.
- Nivida docs on bilateral filtering, `https://docs.nvidia.com/vpi/algo_bilat_filter.html`

Platform Choice

The GHC lab machines and PSC machines provide multi-core CPUs and GPU resources suitable for our parallel implementations. GHC machines will facilitate development and initial testing, while the powerful GPU systems at PSC will enable detailed performance profiling and large-scale experimentation.

Schedule (Tentative)

Week	Planned Activities
3.26 - 4.2	Implement serial version of bilateral filter; begin OpenMP parallelization.
4.2 - 4.9	Analyze CPU performance, identify bottlenecks, and optimize OpenMP parallelization using course techniques.
4.9 - 4.15	Develop initial CUDA implementation; perform GPU profiling and identify bottlenecks.
4.15 - 4.23	Optimize CUDA implementation based on analysis; evaluate GPU performance improvements.
4.23 - 4.28	Finalize benchmarking results, prepare poster and presentation materials.
4.29	Poster session.