HW2 Report

202211169 전상우

1. Implementation Results

image	1024*1024	768*768	512*512	256*256	128*128
Avg speedup	2.368088	2.101901	2.089685	2.155349	2.092644

2. Optimization Approach

A. Block Processing

Processes the image in blocks of size 16x16 pixels.

Cache Efficiency: When, processing data, CPUs work faster if the data is in cache. By processing in blocks, the likelihood that the next needed pixel is already in the cache is increased.

Spatial Locality: This improves spatial locality, which means that memory accesses are more likely to be close to each other, reducing cache misses.

B. Reduced Overhead from Dynamic Memory Allocation

Eliminates dynamic memory allocation, directly calculating and storing results in the pre-allocated output array.

Memory allocation overhead: Functions like ‘malloc’ and ‘free’ have overhead due to their need to manage the heap, search for memory blocks, and handle fragmentation. This overhead is avoided.

Memory fragmentation: Frequent allocations and deallocations can lead to memory fragmentation, making future allocations slower. Avoiding dynamic allocation prevents fragmentation.

C. Improved Memory Access Patterns

Memory accesses are more contiguous and localized within blocks

Cache line utilization: Modern CPUs fetch data in cache lines, which are contiguous blocks of memory. By accessing memory contiguously, the code makes better use of

each cache line fetched.

Reduced Cache misses: Contiguous access patterns reduce the number of cache misses, as the needed data is more likely to be in the cache.

D. Consolidated Boundary Checks

Integrates boundary checks within the main loops and processes pixels in the block

Reduced redundant checks: By moving boundary checks to the outer loops, the code avoids performing the same check multiple times. This reduces the computational overhead.

E. Direct Pixel Manipulation

Computes pixel values directly within the main loop and stores them in the output array.

Function call overhead: Function calls have overhead due to stack management and parameter passing. By computing values directly in the loop, this overhead is eliminated.

Inlined computation: Direct computation within the loop can be more efficiently optimized by the compiler.