Comparison of OpenMP and CUDA Shared Memory Implementations

Overview:

Original Implementation (OpenMP): The original vectorsum code computes an accumulation over 500 iterations. Each iteration multiplies elements of a 1D vector by a loop coefficient and adds the result to an output vector. Two OpenMP methods were explored – parallelizing only the outer loop and collapsing both loops to form a 2D iteration space for improved load balancing. For large vector sizes, collapsing the loops provided better performance (e.g., reducing execution times from around 4-5 seconds in sequential code to roughly 0.2 seconds with 4 threads in optimized builds).

CUDA Shared Memory Implementation: The new CUDA version reinterprets the problem of a 2D vector (matrix) addition using shared memory tiling. The kernel loads tiles of the input arrays into shared memory, performs element-wise addition, and writes the result back. CUDA events are used to time kernel execution. Although the CUDA version demonstrates a simpler element-wise addition compared to the original accumulation, it can be adapted to include coefficient multiplication and accumulation if desired.

Key Changes & Reasoning

1. Platform & Parallel Model:
    a. From CPU to GPU: The OpenMP code runs on multi-core CPUs, whereas the CUDA code exploits thousands of GPU threads for fine-grained parallelism
    b. Kernel vs. Loops: The original nested loops are replaced by a CUDA kernel that executes over a grid of thread blocks, aligning well with the GPU's architecture.
2. Memory Management:
    a. Explicit Data Transfers: CUDA requires explicit allocation and data movement between host and device. This is in contrast to the CPU code, which directly accesses system memory.
    b. Shared Memory and Tiling. The CUDA kernel uses on-chip shared memory to store a tile of data for each block. This reduces global memory access latency and improves throughput. In the OpenMP version, data reuse is limited by the CPU cache, whereas the shared memory in CUDA offers a more predictable performance benefit.

Conclusion:

The CUDA shared memory implementation introduces several fundamental changes– from explicit memory transfers and kernel-based parallelism to optimized shared memory usage–which yield significant speedups on data-heavy tasks. While the original OpenMP code

demonstrates improved performance through multi-threading and loop collapsing, the CUDA version leverages GPU hardware to potentially reduce execution time dramatically for large datasets. This trade-off, however, comes with increased complexity in memory management and kernel design.