P. Hadjidoukas, E. Dermatas

Set 2 - SIMD and CUDA

Issued: January 7, 2026

Question 1: SIMD and WENO5

The fifth-order Weighted Essentially Non-Oscillatory (WENO5) scheme is a high-order numerical method for solving hyperbolic partial differential equations. WENO methods provide high accuracy in smooth regions of the solution while remaining stable and non-oscillatory near discontinuities such as shocks.

In this assignment, you will optimize a reference WENO5 implementation and experimentally evaluate the impact of different vectorization strategies.

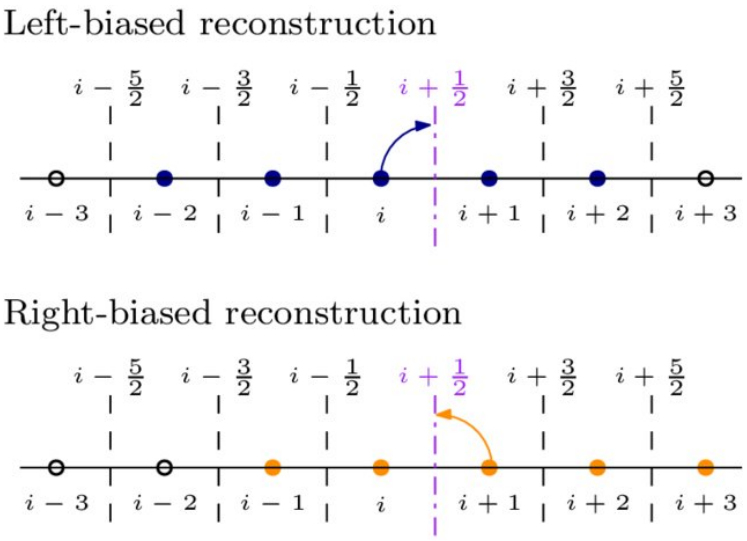


Figure 1: WENO left-biased and right-biased reconstruction stencils

A reference implementation is provided in the `task1` folder. The `bench.c` file contains a benchmarking framework, while `weno.c` includes a baseline scalar implementation of the left-biased reconstruction stencil (hence the suffix `minus`).

a) Extend the code (Makefile and source files) to implement the following optimized versions of the WENO5 kernel:

- **Compiler optimizations + OpenMP SIMD:** Enable appropriate compiler optimizations (e.g., `-O3`, `-march=native`) and implement a vectorized version using `#pragma omp simd`. You may also use OpenMP parallel loops where appropriate.
- **SSE intrinsics:** Provide an explicit vectorized implementation using SSE intrinsics.

- **AVX / AVX2 intrinsics:** Provide an explicit vectorized implementation using AVX (or AVX2 if available) intrinsics.

b) Extend the benchmarking procedure and select an appropriate problem size to experimentally evaluate and compare:

- baseline scalar implementation,
- compiler-optimized + OpenMP SIMD,
- SSE intrinsics,
- AVX / AVX2 intrinsics.

In your report:

- Report wall-clock execution time and **throughput in FLOP/s**. Clearly state the floating-point operation count assumed for the WENO5 kernel.
- Optionally report memory bandwidth (GB/s) and comment on whether the kernel appears compute-bound or memory-bound.
- Discuss the observed performance differences and relate them to vector width, instruction mix, and memory access patterns.

Optional: Apply and report any additional optimizations (e.g., data alignment, padding, loop restructuring, reduced temporaries, or use of `restrict` qualifiers).

Correctness requirement: All optimized implementations must produce results consistent with the reference implementation (within a reasonable floating-point tolerance).

Question 2: CUDA and Complex Matrix Multiplication

Consider two complex matrices of size $N \times N$:

$$(A + Bi), \quad (C + Di),$$

where A, B, C, D are real matrices. Their product is given by:

$$(A + Bi)(C + Di) = (AC - BD) + (AD + BC)i = E + Fi, \tag{1}$$

where E and F are the real and imaginary parts of the result.

a) Develop a CUDA application that computes the matrices E and F. The matrices A, B, C, D are allocated and initialized with random values on the host, copied to the device, and the results are stored in E and F.

b) Implement the complex matrix multiplication using one of the following approaches:

- **cuBLAS-based implementation:** Use cuBLAS GEMM routines to compute the four real matrix products AC, BD, AD, and BC, and then form $E = AC - BD$ and $F = AD + BC$ using CUDA kernels.
- **Custom CUDA kernel:** Implement matrix multiplication directly using CUDA kernels without cuBLAS. In this case, clearly describe your kernel design and optimization choices.

Hint on cuBLAS usage: You are encouraged to consult the official NVIDIA cuBLAS examples when implementing this exercise. In particular, look for examples demonstrating matrix multiplication using `cublasSgemm`, `cublasDgemm`, or `cublasGemmEx`. The NVIDIA *CUDALibrarySamples* repository and the CUDA Toolkit sample `matrixMulCUBLAS` provide minimal, working examples of GEMM usage. Note that cuBLAS assumes column-major (Fortran-style) matrix storage by default; care must be taken when working with row-major (C-style) data layouts.

c) In your report, describe the main design and optimization decisions, including:

- data layout and memory access patterns,
- choice of precision (single vs double),
- kernel configuration (thread blocks, tiling, shared memory usage),
- whether computations are fused or explicitly separated.

d) Measure and report performance for multiple values of N.

- Report wall-clock execution time and **throughput in FLOP/s**. Clearly state the FLOP count used in your performance calculations.
- Optionally compare against a sequential CPU implementation and comment on performance trends and scalability.

Correctness requirement: Validate your results against a CPU reference implementation and report the maximum error for both E and F.