Radix-2 FFT in C++20: Correctness-Centered Design, Mathematical Rigor, and Automated Verification

1st Santiago Florido Gómez
Mechatronic engineer
ENSTA - Paris
Paris, France
santiago.florido@ensta-paris.fr

***Abstract*—This paper presents a complete C++20 implementation of an iterative in-place radix-2 Cooley-Tukey fast Fourier transform (FFT) and its inverse (IFFT), together with a mathematically exact $O(N^2)$ discrete Fourier transform (DFT) reference for validation. The implementation follows a strict engineering sign convention, applies bit-reversal permutation, and includes explicit $1/N$ normalization in the inverse transform. Verification is treated as a primary design objective: automated tests compare FFT output against the direct DFT, validate round-trip stability $\mathrm{IFFT}(\mathrm{FFT}(x)) \approx x$, check pure-tone spectral concentration, and enforce Parseval energy consistency under double precision. This approach yields a dependency-free and reproducible baseline suitable for research prototypes, educational signal-processing laboratories, and production-oriented systems where deterministic spectral behavior and explainable numerical error bounds are required.**

***Index Terms*—fast Fourier transform, mixed-radix, radix-2, AoS, SoA, memory layout, Cooley-Tukey, spectral analysis, numerical verification, C++20**

I. Introduction

The fast Fourier transform (FFT) is one of the most consequential algorithms in computational science because it reduces the complexity of spectral analysis from $O(N^2)$ to $O(N \log_2 N)$ [1]. This complexity reduction is not only asymptotic theory; it determines practical feasibility in systems constrained by real-time deadlines, bounded power budgets, and finite memory bandwidth.

In modern engineering, many physical phenomena are observed or controlled in time or space domains but are most interpretable in the frequency domain. Spectral decomposition is therefore central to communication receivers, radar processing chains, medical imaging reconstruction, vibration diagnostics, acoustic scene analysis, geophysical inversion, and computational physics. In these systems, FFT quality directly influences detection sensitivity, parameter estimation bias, and downstream model robustness.

From a numerical perspective, FFT implementations are also a reliability-critical component. Small indexing errors in butterfly updates, incorrect exponential signs, or misplaced normalization can silently produce spectra that appear plausible yet are physically wrong. Such defects propagate into control loops, inverse problems, and machine-learning feature pipelines. For this reason, an FFT implementation should be delivered with explicit mathematical conventions and independent verification against a direct DFT baseline.

This work addresses that requirement through a C++20, standard-library-only implementation of radix-2 FFT/IFFT and a test methodology that prioritizes reproducibility, deterministic error reporting, and traceable agreement with Fourier definitions [2], [3]. The main contributions are:

- an iterative in-place radix-2 FFT with explicit bit-reversal permutation;
- a definition-aligned $O(N^2)$ DFT/IDFT reference implementation;
- automated validation with strict tolerances for round-trip accuracy, bin-wise agreement, tone localization, and energy conservation.

The resulting project forms a rigorous baseline that can be extended toward optimized kernels, mixed-radix strategies, and architecture-specific acceleration while preserving a verifiable correctness core.

II. Mathematical Foundations

A. DFT and IDFT Definitions

For a length-N complex sequence $x[n]$, the forward transform is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \quad k = 0, \ldots, N-1, \tag{1}$$

and the inverse transform is

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{+j2\pi kn/N}, \quad n = 0, \ldots, N-1. \tag{2}$$

The implementation follows this sign convention exactly and applies normalization only in the inverse path.

B. Roots of Unity and Notation

Define

$$W_N = e^{-j2\pi/N}, \tag{3}$$

so that $W_N^{kn} = e^{-j2\pi kn/N}$. Equation (1) becomes

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{kn}. \tag{4}$$

This notation enables compact derivation of radix-2 factorization.

C. Radix-2 Cooley-Tukey Decomposition

Assume $N = 2^m$. Split the sequence into even and odd indices:

$$X[k] = \sum_{r=0}^{N/2-1} x[2r] W_N^{k(2r)} + \sum_{r=0}^{N/2-1} x[2r+1] W_N^{k(2r+1)}. \tag{5}$$

Using $W_N^2 = W_{N/2}$, define

$$E[k] = \sum_{r=0}^{N/2-1} x[2r] W_{N/2}^{kr}, \quad O[k] = \sum_{r=0}^{N/2-1} x[2r+1] W_{N/2}^{kr}, \tag{6}$$

then

$$X[k] = E[k] + W_N^k O[k], \tag{7}$$
$$X[k + N/2] = E[k] - W_N^k O[k]. \tag{8}$$

Equations (7) and (8) are the radix-2 butterfly relations.

D. Bit-Reversal Permutation

In an iterative decimation-in-time schedule, input indices are permuted by reversing their m-bit binary representation before stage-wise butterflies are applied. This permutation maps recursive subproblem order into contiguous in-place blocks. Without it, stage-local butterfly connectivity is violated and final bins are scrambled.

E. Complexity

A direct DFT computes N outputs, each with N terms, yielding $\Theta(N^2)$. The radix-2 FFT performs $\log_2 N$ stages with $\Theta(N)$ work per stage, yielding $\Theta(N \log_2 N)$. This is the computational foundation for real-time spectral systems.

III. Algorithm and Implementation Mapping

The codebase evaluates multiple transform models under one interface, including iterative radix-2 FFT, iterative mixed-radix (4/2) FFT, recursive radix-2 FFT, split-radix FFT, direct $O(N^2)$ DFT reference, and new SoA layout counterparts for radix-2 and mixed-radix. This section details the mapping used by the radix-2 iterative baseline, while mixed-radix and split-radix decompositions are detailed in Sections IV and V, respectively.

A. Iterative In-Place Kernel

The implementation processes stages with block lengths $L = 2, 4, \ldots, N$. For each block start index i and local offset $j \in [0, L/2 - 1]$:

$$u = x[i + j], \tag{9}$$
$$v = x[i + j + L/2]\, W_L^j, \tag{10}$$

then updates

$$x[i + j] \leftarrow u + v, \tag{11}$$
$$x[i + j + L/2] \leftarrow u - v. \tag{12}$$

This mapping is a direct implementation of the butterfly equations.

B. Twiddle Generation Strategy

At each stage, the principal stage twiddle $W_L = e^{\pm j2\pi/L}$ is computed once. Per-butterfly twiddles are obtained by recurrence $w \leftarrow wW_L$ within each block. This avoids storing global twiddle tables while preserving deterministic behavior and numerical clarity.

C. Inverse Transform

The inverse transform reuses the same kernel with opposite sign in the exponential and applies a final scaling by $1/N$. This ensures consistency with (2) and supports stable round-trip reconstruction.

D. Input Validation

Because the chosen factorization is radix-2, valid sizes must satisfy $N \neq 0$ and N power-of-two. Invalid sizes throw `std::invalid_argument` with an explicit message, preventing undefined spectral behavior.

E. Implementation Notes

The code is written in C++20 with only the standard library, using `std::complex<double>` for numerical representation. The design goal is not architecture-specific peak throughput but reference-quality correctness that remains portable across Linux and Windows toolchains.

IV. Iterative Mixed-Radix (4/2) FFT

The implemented mixed-radix transform targets lengths $N = 2^m$ and is strictly iterative and in-place. The stage plan is defined by the parity of m:

- if m is even, all stages are executed as radix-4 equivalents;
- if m is odd, one radix-2 stage is executed first, followed by radix-4 equivalents.

The transform uses the same Fourier-sign convention as the other FFT models:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \tag{13}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{+j2\pi kn/N}. \tag{14}$$

Accordingly, the inverse applies a final $1/N$ scaling.

A. Ordering Strategy and Equivalence

The data are first permuted by binary bit-reversal, exactly as in the iterative radix-2 baseline. This is essential: preserving the same input permutation guarantees the same output indexing once each processing stage is algebraically equivalent to the corresponding radix-2 stages.

Each radix-4 stage in this implementation is not introduced as a separate digit-reversal pipeline. Instead, it is constructed as a fusion of two consecutive radix-2 stages. Therefore, for every fused step, the resulting mapping is mathematically identical to applying those two radix-2 stages in sequence; only the computational organization differs.

B. Two-Stage Fusion Formula

Let L be the butterfly length of the first stage in a fused pair, and define $W_L = e^{-j2\pi/L}$, $W_{2L} = e^{-j2\pi/(2L)}$ for the forward transform. For an in-block frequency index $p \in [0, L/2-1]$, first-stage combinations are

$$A_0 = U_0 + W_L^p U_1, \qquad A_1 = U_0 - W_L^p U_1, \tag{15}$$

$$B_0 = V_0 + W_L^p V_1, \qquad B_1 = V_0 - W_L^p V_1. \tag{16}$$

The second stage of length $2L$ then yields

$$Y_0 = A_0 + W_{2L}^p B_0, \qquad Y_2 = A_0 - W_{2L}^p B_0, \tag{17}$$

$$Y_1 = A_1 + \rho\, W_{2L}^p B_1, \qquad Y_3 = A_1 - \rho\, W_{2L}^p B_1, \tag{18}$$

with

$$\rho = \begin{cases} -j, & \text{forward FFT,} \\ +j, & \text{inverse FFT.} \end{cases}$$

The ρ factor corresponds to a quarter-turn phase shift and is the key mixed radix-4/2 coupling term in the fused kernel.

C. Twiddle Evaluation Policy

To avoid expensive transcendental evaluations in the innermost butterfly loop, each stage computes only principal roots once, then obtains all required powers by multiplicative recurrence. This preserves numerical consistency while reducing loop overhead and instruction latency.

D. Complexity and Practical Implications

For $N = 2^m$, the number of computational stages is

$$\begin{cases} m/2, & m \text{ even,} \\ 1 + (m-1)/2, & m \text{ odd,} \end{cases}$$

where each fused stage represents two radix-2 levels. The asymptotic complexity remains $O(N \log_2 N)$, but constant factors can improve due to stage fusion and reduced memory traffic relative to executing two separate radix-2 passes.

V. Split-Radix FFT

Split-radix is a structured Cooley-Tukey factorization that combines radix-2 and radix-4 style decomposition in one recursion. For a power-of-two size N, the transform is split into one half-size branch and two quarter-size branches:

$$\mathrm{DFT}(N) \rightarrow \mathrm{DFT}(N/2) + \mathrm{DFT}(N/4) + \mathrm{DFT}(N/4). \tag{19}$$

The even-indexed samples feed the $N/2$ branch. The odd-indexed samples are separated into two classes, $n = 4m+1$ and $n = 4m+3$, each producing one $N/4$ branch. In the implementation, these two odd branches are recombined with twiddle factors and $\pm j$ rotations to recover the four output quadrants.

Compared to pure radix-2, split-radix reduces arithmetic operation count in theory, especially in multiplication count for large transforms. This makes it a relevant candidate when algorithmic operation complexity is the primary optimization target.

The trade-off is higher implementation complexity. Index mapping is less uniform than standard radix-2 butterflies, and memory access patterns can be less cache-friendly depending on recursion depth, temporary-buffer layout, and compiler optimization. Consequently, wall-clock speedup is architecture-dependent and must be validated experimentally rather than assumed from operation counts alone.

VI. Memory Layout Optimization: AoS vs SoA

A. Motivation and Controlled Scope

The primary benchmark in Section 3 compares algorithmic families. This section adds a second, controlled experiment focused only on memory layout for the two strongest iterative baselines: radix-2 and mixed radix (4/2).

AoS and SoA represent two distinct organizations for complex signals:

$$\text{AoS: } x[n] = \Re\{x[n]\} + j\,\Im\{x[n]\}, \tag{20}$$

$$\text{SoA: } \mathbf{r}[n] = \Re\{x[n]\}, \quad \mathbf{i}[n] = \Im\{x[n]\}. \tag{21}$$

The compared variants are `radix2_aos`, `radix2_soa`, `mixed42_aos`, and `mixed42_soa`.

B. Mathematical Equivalence Under Layout Change

The FFT operator is unchanged:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi kn/N}, \quad x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{+j2\pi kn/N}. \tag{22}$$

Only storage and arithmetic organization differ.

For a generic butterfly with $u = u_r + ju_i$, $t = t_r + jt_i$, and twiddle $w = w_r + jw_i$,

$$v = t\,w, \tag{23}$$

$$\Re\{v\} = t_r w_r - t_i w_i, \tag{24}$$

$$\Im\{v\} = t_r w_i + t_i w_r, \tag{25}$$

followed by

$$y_0 = u + v, \tag{26}$$

$$y_1 = u - v. \tag{27}$$

AoS computes these relations on interleaved complex objects; SoA computes the same equations on two contiguous real streams. Therefore, spectral outputs are mathematically equivalent up to floating-point roundoff.

C. *Performance Rationale*

In AoS, real and imaginary fields are interleaved in memory. In SoA, each field is contiguous, which reduces address-stride irregularity in butterfly loops and can improve cache-line utilization and compiler scheduling. Even without explicit SIMD intrinsics, SoA often provides a better substrate for contiguous loads and future vectorized kernels.

D. *SIMD Integration in the SoA Kernels*

The SoA implementations of both iterative radix-2 and iterative mixed radix (4/2) now include explicit SIMD execution paths. The runtime dispatcher selects the widest available vector ISA in the order AVX-512, AVX2, then scalar fallback.

For radix-2, each stage precomputes the twiddle sequences $\{w_r[j]\}$ and $\{w_i[j]\}$ and applies butterflies in vector blocks across contiguous index ranges j. For a vector lane set $\mathcal{J}$, the kernel evaluates

$$\mathbf{v}_r = \mathbf{o}_r \odot \mathbf{w}_r - \mathbf{o}_i \odot \mathbf{w}_i, \tag{28}$$

$$\mathbf{v}_i = \mathbf{o}_r \odot \mathbf{w}_i + \mathbf{o}_i \odot \mathbf{w}_r, \tag{29}$$

followed by vector add/subtract updates for the even and odd outputs. Here $\odot$ denotes element-wise vector multiplication.

For mixed radix (4/2), the fused radix-4 stage uses the same vectorized complex arithmetic for the three twiddle families required by the decomposition: W_L^j, W_{2L}^j, and jW_{2L}^j (with sign adapted for inverse transform). The four output branches of each fused butterfly are then updated with vector add/subtract combinations of (p_0, p_1, t_0, t_1), preserving the exact scalar formulation.

Inverse transforms retain the same normalization convention by applying a SIMD vector scaling by $1/N$ after stage completion.

E. *Layout-Study Figures*

Figures 1 and 2 isolate layout effects by keeping transform definition, normalization, benchmark sizes, warmup policy, and measured iterations fixed. This allows direct attribution of runtime differences to memory organization rather than algorithmic reformulation.

VII. Verification Methodology

A. *Automated Test Set*

Verification is executed through CTest and a self-contained test executable. The FFT-oriented checks are applied to all power-of-two FFT models, including AoS and SoA variants of iterative radix-2 and iterative mixed-radix (4/2), plus recursive

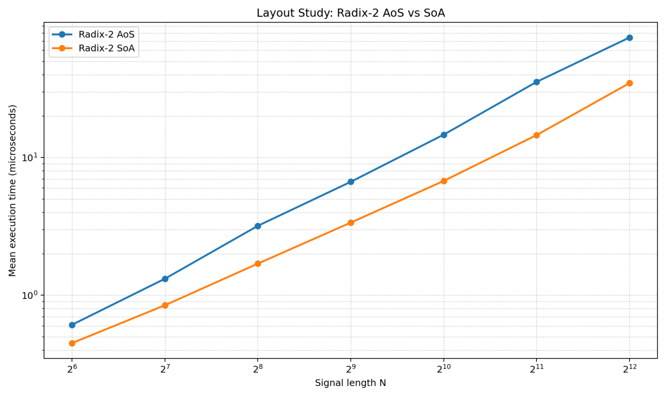

Fig. 1. Radix-2 runtime comparison under AoS and SoA memory layouts.

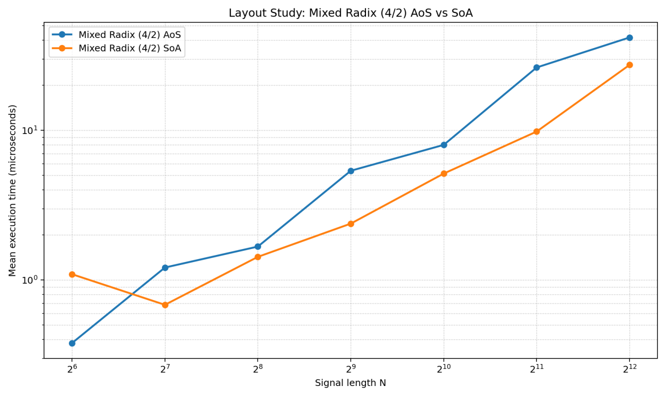

Fig. 2. Mixed radix (4/2) runtime comparison under AoS and SoA memory layouts.

radix-2 and split-radix. The direct DFT model is validated through dedicated checks. The suite includes:

- **Round-trip stability:** for powers of two from $N = 2$ to $N = 4096$, verify $\text{IFFT}(\text{FFT}(x)) \approx x$ using random complex vectors with fixed seed.
- **Reference agreement:** for $N \leq 256$, compare each FFT bin against the direct $O(N^2)$ DFT.
- **Layout consistency:** verify that SoA outputs match their AoS counterparts for the same input and ordering convention.
- **Pure-tone localization:** for $x[n] = e^{j2\pi kn/N}$, verify dominant energy at bin k and near-zero leakage elsewhere.
- **Parseval consistency:** verify

$$\sum_n |x[n]|^2 \approx \frac{1}{N} \sum_k |X[k]|^2, \tag{30}$$

consistent with the chosen normalization.
- **Input constraints:** verify rejection of non-power-of-two sizes.

B. *Tolerances and Rationale*

Double-precision tolerances are set to 10^{-10} for round-trip relative L_2 error, bin-wise FFT-vs-DFT absolute error, and

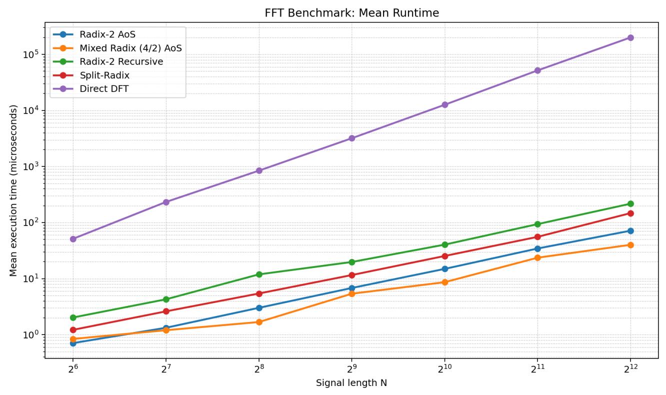

Fig. 3. Average execution time as a function of signal length N for the primary algorithmic study.

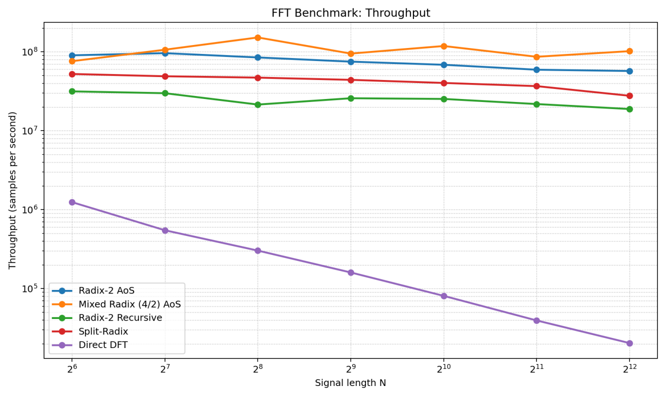

Fig. 4. Throughput as a function of signal length N for the primary algorithmic study.

Parseval relative mismatch. A 10^{-8} leakage bound is used for off-peak bins in pure-tone tests. These values reflect expected floating-point accumulation and trigonometric evaluation error under IEEE-754 arithmetic [5].

C. Expected Spectral Behavior

A single-tone complex exponential is expected to produce one dominant peak at the target bin k. A real sine wave is expected to produce two symmetric dominant peaks at k and $N - k$, consistent with conjugate symmetry for real-valued time signals.

VIII. Benchmark Results and Comparison

This section reports the primary algorithmic comparison study. The evaluated models are: `radix2_aos`, `mixed42_aos`, `radix2_recursive`, `split_radix`, and `direct_dft`. The benchmark uses $N \in \{64, 128, 256, 512, 1024, 2048, 4096\}$, 5 warmup iterations, 40 measured iterations, and a fixed seed.

The reported figures summarize:

- transform length N;
- average runtime per transform;
- throughput in processed samples per second.
- Additional descriptors are also measured in the pipeline: median runtime, minimum and maximum runtime, runtime dispersion, high-percentile latency, and normalized costs per sample and per $N \log_2 N$.

Figure 3 shows the expected asymptotic separation between FFT factorizations and direct DFT. The direct $O(N^2)$ model becomes rapidly dominant in runtime as N increases, while all FFT variants preserve the $O(N \log_2 N)$ trend.

Figure 4 presents the same behavior in throughput units. Iterative implementations remain the strongest practical baseline in this environment, and recursive decompositions remain useful as algorithmic references but with lower effective throughput due to call and traversal overhead.

This primary study provides the baseline against which the dedicated memory-layout study in Section VI is interpreted.

IX. Scientific and Engineering Relevance

FFT is foundational in projects where information is encoded in spectral structure rather than raw samples. In digital communications, orthogonal frequency-division multiplexing (OFDM) modulators and demodulators rely on FFT/IFFT pairs for subcarrier-domain processing and channel equalization. In radar and sonar, FFT stages support range-Doppler processing and clutter separation. In biomedical instrumentation, electroencephalography and electrocardiography pipelines use spectral features for diagnosis and anomaly detection.

In imaging and inverse problems, FFT-based convolution accelerates iterative solvers for magnetic resonance reconstruction, computational microscopy, and deblurring. In mechanical and civil monitoring, vibration spectra expose resonance modes and early fault signatures in rotating machinery and structures. In power systems, harmonic analysis depends on FFT stability to quantify distortion and nonstationary events.

For machine-learning systems, FFT-derived features are frequently used in audio classification, activity recognition, and predictive maintenance. In these data-centric workflows, a well-verified FFT implementation reduces silent label noise caused by spectral misalignment and improves reproducibility across hardware and operating systems.

Therefore, FFT is not only an algorithmic acceleration; it is a scientific instrumentation primitive. Correctness, traceable conventions, and controlled numerical error are mandatory for high-impact engineering projects.

X. Conclusion

This work provides a correctness-verified FFT framework in C++20 with two complementary benchmark layers. The primary layer compares algorithmic families (iterative radix-2, mixed radix (4/2), recursive radix-2, split-radix, and direct DFT). The secondary layer isolates memory-layout effects (AoS versus SoA) for the two strongest iterative families.

The primary study confirms the expected complexity separation: FFT factorizations preserve scalable behavior while direct DFT becomes rapidly impractical as N grows. The layout study then shows how data organization alone can shift runtime for a fixed transform definition and fixed benchmark protocol.

Overall, the framework now supports both algorithm-level and memory-level optimization analysis under a single reproducible pipeline. This structure is suitable for subsequent work on explicit SIMD kernels, planner-based execution strategies [4], and embedded performance portability while preserving mathematical traceability and automated verification.

Acknowledgment

This work did not receive external funding.

References

[1] J. W. Cooley and J. W. Tukey, ”An algorithm for the machine calculation of complex Fourier series,” *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[2] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1999.

[3] R. N. Bracewell, *The Fourier Transform and Its Applications*, 3rd ed. New York, NY, USA: McGraw-Hill, 2000.

[4] M. Frigo and S. G. Johnson, ”The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, Feb. 2005.

[5] IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019, 2019.