Literature Review: Progression of the Use of High-Performance Computing for FFTs

By: Shree Singhal

1 Introduction

The Fast Fourier Transform (FFT) is one of the most consequential algorithms devised as it allows for signals to be decomposed into their simpler components. Devised by Cooley and Tukey in 1965, FFT made it possible to analyze the frequency spectrum of a signal on digital hardware, and it became a foundational building block of modern signal processing [13]. Six decades later in 2026, the FFT continues to support a wide range of computationally intensive workloads, including molecular dynamics, turbulence simulation, medical imaging, radar processing, and wireless communications among others. In some scientific applications, the FFT consumes as much as 70% of total runtime, making its performance a direct determinant of overall application throughput [9]. Consequently, extensive research has been conducted in recent years to optimize FFT algorithms for high-performance computing (HPC) systems.

Modern simulations routinely require three-dimensional FFTs on grids of 4096^3 or larger, and exascale-class scientific codes have driven this scale even higher [6]. At the same time, signal-processing applications such as radar, computational MRI, and 6G baseband processing require near real-time latencies that single-core implementations cannot meet. These two issues, the growing problem size and tightening latency requirements, have made parallelism a fitting solution. However, the FFT presents a particularly difficult challenge for parallel architectures: with only O(N log N) floating-point operations to perform on N data points, it is a memory and communication-bound algorithm rather than a compute-bound one [5]. Researchers working on the Summit pre-exascale system have reported that distributed 3D FFTs spend more than 97% of their runtime in MPI communication when scaled across many GPU-equipped nodes, leaving compute units largely idle [5]. This bottleneck has prompted substantial research in the field to find solutions.

The Discrete Fourier Transform of a sequence of N complex samples f[n] is shown in Equation 1.

$$\mathrm{F(m)} = \sum_{n=0}^{\mathrm{N-1}} \mathrm{f}(n) e^{-i2\pi \frac{nm}{2BL}} = \sum_{n=0}^{\mathrm{N-1}} \mathrm{f}(n) e^{-i2\pi \frac{nm}{N}}$$

Equation 1: DFT

Each output F[m] is a weighted sum of all N input samples, with the complex exponentials e^(−i2πnm/N). Computed directly, this expression requires $O(N^2)$ complex multiplications and additions. The Cooley–Tukey FFT algorithm improves upon this by recursively decomposing an N-point DFT into smaller DFTs, reducing the operation count to O(N log N) [13]. The total number of arithmetic operations is roughly $5N \log_2 N$ for a complex-valued radix-2 FFT. This is a relatively low complexity; an $N = 2^{20}$ FFT requires about 10^8 floating-point operations, which a

single GPU streaming multiprocessor can finish in milliseconds [14]. Therefore, the difficulty is not the arithmetic itself but the data-access pattern that produces it. Each butterfly stage reads pairs of values that are separated by a stride that doubles at every level of the recursion, so the algorithm sweeps through the entire data array $\log_2 N$ times with large stride patterns that prevent implementing optimizing features like cache locality and coalesced memory access [2][1]. The arithmetic intensity (ratio of floating-point operations to bytes moved) is therefore very low: O(log N) operations per word transferred. This is far below the roughly 13,440 operations per FP64 word required to keep a HPC Summit node's compute units saturated [5]. Additionally, when the transform is spread across many nodes, a multi-dimensional FFT decomposed over a processor grid requires a global all-to-all transpose between the one-dimensional FFTs along each axis; as a result, the collective communication must transfer every input value across the interconnect at least once.

As a result, large scale FFT reduces, in Dr. Jack Dongarra's words, "to a communications problem" [5][17]. Single-node DSP implementations on embedded processors or FPGAs have achieved high throughput for small transform sizes, and there is already substantial research on such designs [15][16]. However, this paper will focus on simulation and data-analysis workloads whose problem sizes, data volumes, and/or turnaround requirements necessitate parallel execution. Already implemented exascale science applications include the LAMMPS molecular-dynamics package, the HACC cosmology code, and the QMCPACK quantum-simulation suite. All of them rely on distributed three-dimensional FFTs as a critical performance-determining kernel. Additionally, the U.S. Department of Energy's Exascale Computing Project identified more than a dozen of its target applications as FFT-dependent [5][18].

For context for the following discussion, this is an explanation of slab vs pencil decomposition. Distributed FFT implementations must choose how to split the 3D data grid across processors. Slab decomposition divides the grid along a single axis, giving each processor a flat slab of data; this requires only one transpose between FFT stages but limits the implementation to at most N processors for an N^3 grid. Pencil decomposition divides the grid along two axes simultaneously, giving each processor a long thin pencil of data; this raises the scalability max to N^2 processors at the cost of requiring two transposes per FFT. Each transpose corresponds to an all-to-all communication operation across the processor grid, so the choice between slab and pencil is a tradeoff between scalability and communication volume. Some of the papers surveyed in this review take varying positions on this tradeoff depending on their target hardware: libraries built for petascale-and-beyond core counts adopt pencil decomposition, while libraries targeting dense GPU systems with comparatively modest accelerator counts often return to slab decomposition.

This paper will highlight the progression of high-performance computing techniques for Fast Fourier Transforms from the 20th century to present day. The following section will summarize papers in chronological order of the year published, and the section thereafter will assess the connections between papers.

2 Summary of Papers Being Compared

2.1 Frigo & Johnson (2005): The Design and Implementation of FFTW3 [1]

Frigo and Johnson's paper introduces FFTW3, the third major version of the Fastest Fourier Transform in the West library. It demonstrates that no single FFT implementation is optimal across the full range of modern computing hardware. The best algorithm depends on transform size, processor architecture, cache hierarchy, and the availability of SIMD vector units. Therefore, a portable FFT software must select its algorithm at runtime through a process the authors call autotuning. Also, it introduces a code-generation infrastructure called *genfft* that produces small, highly optimized FFT kernels (the authors called them "codelets") which the autotuner composes into larger transforms. This combination of runtime planning and offline code generation was later adopted in in BLAS implementations including ATLAS, OpenBLAS and GPU libraries including the primary FFT library cuFFT.

The paper begins by explaining the Cooley–Tukey decomposition. For an N-point DFT where $N = N_1N_2$, the algorithm reorganizes the transform as N_1 smaller DFTs of size N_2, followed by element-wise multiplication by twiddle factors, followed by N_2 smaller DFTs of size N_1. This exposes the choices that can be made, which are the radix at each level of recursion, the order in which sub-transforms are computed, the data layout in memory, and whether and how to vectorize across SIMD lanes. The FFTW3 planner explores this space to identify the fastest plan for a given transform size on the target machine. The resulting plan is then executed on the actual data, often many times in scientific applications where the same transform is performed repeatedly on different inputs.

The authors report that FFTW3 outperforms or matches every contemporary FFT library across a comprehensive benchmark suite spanning transform sizes from 2 to 2^{20}, multiple architectures (Intel x86, IBM POWER, SPARC, etc.), and both single- and double-precision arithmetic. This paper was listed first because it gives the foundational understanding of many later contributions, since each later paper either extend FFTW's autotuning approach to a new architecture or declare a more specialized goal.

2.2 Govindaraju et al. (2008): High Performance Discrete Fourier Transforms on Graphics Processors [2]

Govindaraju and colleagues' SC '08 paper established a GPU as a viable HPC platform for FFT computation. It is notable that this was published at a time when GPU programming was transitioning from graphics-shader-based approaches to general-purpose programming models.

The authors implement and evaluate FFTs on the NVIDIA GeForce 8800 GTX, a GPU with 128 scalar processors organized into 16 streaming multiprocessors and a peak memory bandwidth of 86 GB/s. Their methodology centers on three architectural observations. First, GPU global memory accesses are far more expensive than arithmetic, so the FFT's stride-doubling memory access pattern becomes a bottleneck on GPUs, where memory access must be coalesced across thread groups. Second, GPUs have small but very fast on-chip shared memory (16 KB per multiprocessor on the 8800 GTX), and FFT performance depends on staging data through this shared memory to avoid repeated global-memory traffic. Third, the GPU's parallel execution model rewards algorithms with large numbers of independent threads and uniform control flow, which the FFT's regular butterfly structure provides if the implementation is structured correctly. The authors apply these principles by implementing a hierarchical FFT that performs small stage groups all within shared memory before writing intermediate results back to global memory, and by using GPU texture memory to cache twiddle factor lookups.

On the 8800 GTX, the authors achieve sustained FFT performance of roughly 50 to 80 GFLOP/s for one-dimensional power-of-two transforms in the size range 2^8 to 2^{24}, compared to roughly 4 to 7 GFLOP/s for FFTW3 running on a contemporary 2.4 GHz Intel Core 2 Duo CPU. The speedups were approximately 10x over the previously best CPU implementation. They correspond with roughly 50-60% of the GPU's peak memory bandwidth, confirming the paper's central claim that the FFT on GPUs is bandwidth-limited rather than compute-limited. The authors also evaluate two-dimensional and three-dimensional transforms, where their implementation again outperforms FFTW3 by an order of magnitude, and they compare against an early version of NVIDIA's cuFFT library, demonstrating that their hand-optimized implementation matched or exceeded the vendor library on most transform sizes at the time.

The authors reach the FFT's performance behavior into memory bandwidth, shared-memory utilization, and instruction-throughput components, providing a model that subsequent GPU FFT work has refined but not fundamentally replaced. The work targeted a single-GPU, single-node configuration and does not address the multi-GPU or multi-node scaling. This would be the next step carried out in future research papers in the field.

2.3 Czechowski et al. (2012): On the Communication Complexity of 3D FFTs and its Implications for Exascale [3]

While Govindaraju et al. analyzed the FFT's memory behavior on a single GPU, Czechowski et al. analyzes it in multi-node settings, asking how much data movement is fundamentally required when a 3D FFT is distributed across thousands of processors. The paper's central contribution is a set of communication lower bounds for 3D FFTs that any distributed implementation must meet or exceed, along with a model that predicts how these bounds will scale on exascale-class systems.

The authors derive lower bounds on the volume of data that must be moved between processors during a 3D FFT computation. Their analysis considers both the standard slab decomposition (in which the 3D grid is divided into 2D slabs along one axis) and the pencil decomposition (in which the grid is divided into 1D pencils along two axes). For an N^3 grid distributed across P processors, they show that the slab decomposition requires $\Omega(N^3/P)$ words to be communicated per processor but limits scalability to $P \leq N$ processors, while the pencil decomposition lifts this scalability ceiling to $P \leq N^2$ at the cost of higher per-processor communication volume. They then construct a performance model that combines these communication bounds with realistic projections of exascale-system parameters such as interconnect bandwidth, latency, and per-node compute throughput. The model is parameterized using measurements from contemporary leadership-class systems, allowing the authors to extrapolate forward to projected exascale machines.

The authors predict that, on exascale systems, distributed 3D FFTs would spend most of their runtime in interconnect communication rather than in floating-point arithmetic, with communication accounting for 90% or more of total runtime for typical problem sizes. The authors also demonstrate that the asymptotic gap between slab and pencil decompositions gets smaller at large processor count.

The communication lower bounds are derived using established techniques from the communication-avoiding algorithms literature and are therefore mathematically defensible, while the performance model grounds these bounds in measured hardware parameters.

Where Govindaraju et al. established that FFTs on a single GPU are memory-bandwidth-limited, Czechowski et al. establish that FFTs across many nodes are interconnectbandwidth limited, and they provide the theoretical apparatus needed to reason about both.

2.4 Pekurovsky (2012): P3DFFT — A Framework for Parallel Computations of Fourier Transforms in Three Dimensions [4]

Pekurovsky's paper introduces P3DFFT (Parallel Three-Dimensional Fast Fourier Transform). Pekurovsky's work demonstrated how a production library can overcome the scalability bottleneck of one-dimensional decomposition by implementing two-dimensional pencil decomposition in practice. P3DFFT has since been used in turbulence simulations, climate modeling, astrophysics, and material-science codes.

The library uses pencil decomposition, in which the 3D data grid is divided into rectangular columns rather than slabs. For a problem distributed across P processors, the processors are arranged in a virtual two-dimensional grid of dimensions $M_1 \times M_2$, where $M_1 \times M_2 = P$, and each processor is responsible for a pencil-shaped subdomain of the data. The 3D FFT is then computed in three compute stages, corresponding to one-dimensional FFTs along the X, Y, and Z axes, separated by two parallel transpose operations that rearrange the data so that each one-dimensional FFT operates on data that is local to a single processor. Each transpose is implemented as an all-to-all exchange within either a row or column of the virtual processor grid, rather than across all processors. This restriction to subgroups of processors is critical for scalability, because it limits the number of processors involved in any single collective communication operation. P3DFFT relies on existing one-dimensional FFT libraries (FFTW or IBM ESSL) for the local computations, and its own contribution is the orchestration of data decomposition, transposition, and communication.

The paper's result is that P3DFFT achieved 45% weak-scaling efficiency from 128 to 65,536 cores on a Cray XT5 system for a 3D FFT of size 8192^3. This is a substantial because it maintained nearly half of ideal scaling efficiency across a 512 times increase in processor count places. Pekurovsky also reports detailed comparisons between 1D (slab) and 2D (pencil) decompositions on the same hardware, confirming the theoretical prediction that slab decomposition is faster at modest scales (roughly $P \leq N$ processors, where only one transpose is required instead of two) but that pencil decomposition becomes essential beyond that threshold. The library further supports a few practically useful features: real-to-complex and complex-to-real transforms, Chebyshev and sine/cosine transforms in addition to standard Fourier transforms, in-place and out-of-place execution, and uneven data grids in which the problem size is not evenly divisible by the processor grid dimensions.

P3DFFT's was also uniquely production-ready. Unlike research-prototype libraries, P3DFFT was designed for portability across architectures, was tested on a variety of systems, and was released as open-source software with documentation, sample programs, and a stable API. This has resulted in adoption by many real scientific applications. Additionally, the pencil decomposition's two transpose operations introduce communication overhead that more recent libraries have attempted to reduce through techniques such as overlapping computation with communication, hybrid 1D/2D decompositions, and reduced-precision communication [14].

2.5 Ayala et al. (2020): heFFTe — Highly Efficient FFT for Exascale [5]

Ayala's ICCS 2020 paper introduces heFFTe (Highly Efficient FFT for Exascale, pronounced "hefty"), a distributed FFT library developed under the U.S. Department of Energy's Exascale Computing Project (ECP). The paper addresses a gap that had emerged in the distributed FFT software landscape: existing libraries such as P3DFFT, FFTMPI, and SWFFT were either CPU-only or did not scale efficiently on the heterogeneous GPU-accelerated nodes that were expected to dominate exascale-class systems. The authors argue that simply enhancing classical FFT libraries with autotuning or code generation had proven insufficient to deliver high performance on emerging exascale hardware, and they propose that distributed FFT library design be revisited from the ground up with GPU-accelerated heterogeneous architectures as a primary target. The paper presents the design, implementation, and performance evaluation of heFFTe on the Summit supercomputer at Oak Ridge National Laboratory.

The library follows the pencil decomposition approach popularized by P3DFFT but introduces several innovations targeted at GPU-equipped systems. heFFTe supports both binary point-to-point and collective MPI communication primitives, allowing the library to select the most efficient communication pattern for a given problem size and processor configuration. The authors implement custom data-packing and unpacking kernels that prepare buffers for MPI transfer, with multiple options that can be selected through autotuning at runtime. To accelerate the local one-dimensional FFTs that occur between transpose stages, heFFTe interfaces with vendor libraries (FFTW on CPUs, cuFFT on NVIDIA GPUs, rocFFT on AMD GPUs) and combines this with GPU-aware MPI implementations that perform GPU-to-GPU data transfers without staging through CPU memory. The library also supports a slab decomposition mode for cases in which the processor count is small enough to make the slab approach's reduced communication count favorable. The API is modeled on the conventions of FFTW3 and cuFFT to lower the adoption barrier for application developers, and provides templates for multiple data types and precisions.

On Summit, heFFTe's GPU implementation achieves over 40 time speedup on local FFT computations relative to the CPU-only baseline, and over 2 time speedup on the complete distributed 3D FFT computation when compared against FFTMPI and SWFFT, the libraries used by ECP applications such as LAMMPS and HACC. The authors report strong scalability for a 1024^3 FFT and weak scalability for increasing problem sizes, achieving near-linear scaling up to 24,576 IBM Power9 cores and 6,144 NVIDIA V100 GPUs. A particularly important finding is that on the GPU configuration, MPI communication consumed more than 97% of total runtime, while on the CPU configuration the split between local computation and communication was roughly 50/50. This result quantifies how dramatically GPU acceleration shifts the FFT performance bottleneck: the local computation accelerates so much that the interconnect becomes the dominant cost. The paper also includes a roofline analysis showing that heFFTe approaches the achievable peak for the communication-bound regime, confirming that further significant speedups will require improvements to the communication framework rather than to local kernels.

2.6 Verma et al. (2022): Scalable Multi-node Fast Fourier Transform on GPUs [6]

Verma and colleagues present a custom multi-node, multi-GPU FFT implementation built specifically for NVIDIA's Selene supercomputer, an internal NVIDIA system constructed from

DGX A100 nodes connected by Mellanox InfiniBand HDR networks. The authors deliberately position their work against existing distributed FFT libraries, including heFFTe, by arguing that general-purpose libraries inherit design constraints from their CPU-era origins that are no longer well-matched to systems where individual nodes contain eight A100 GPUs interconnected by NVLink. Their goal is maximum throughput on a specific class of dense GPU systems, and the paper documents how that focused design choice translates into measured performance gains.

The implementation departs from the pencil decomposition, which has been the primary method used in distributed FFT libraries since P3DFFT, returning instead to slab decomposition. The authors justify this choice through a direct application of the scaling argument: slab decomposition is limited to $P \leq N$ processors, but on GPU-equipped systems the number of available accelerators is typically far smaller than the linear grid dimension, so the scalability ceiling is rarely reached in practice. Slab decomposition requires only one all-to-all transpose per FFT instead of the two required by pencil decomposition, halving the dominant communication cost. The local one-dimensional FFTs are performed using cuFFT, and the all-to-all transpose is implemented using NCCL (the NVIDIA Collective Communication Library) rather than MPI. NCCL is designed specifically for GPU-to-GPU communication and exploits NVLink topology within nodes and InfiniBand between them, bypassing the staging overhead that GPU-aware MPI implementations sometimes introduce. The authors also implement an asynchronous communication scheme in which packing of outgoing data and unpacking of incoming data are overlapped with the local FFT computations through careful use of CUDA streams.

Performance results are reported for 3D FFTs of size up to 4096^3 scaling across 32 DGX A100 nodes, or 256 A100 GPUs in total. For a 1024^3 transform, the authors report approximately 6.7 milliseconds on 64 GPUs, achieving roughly 4 time speedup over published heFFTe results on equivalent hardware. For the largest configuration tested, 4096^3 on 256 GPUs, the implementation completes a forward FFT in approximately 84 milliseconds. Strong scaling efficiency is reported as approximately 75% from 8 to 64 GPUs and decreases to roughly 50% at 256 GPUs, consistent with the increasing dominance of inter-node communication at higher scales. The authors also show that intra-node NVLink communication accounts for approximately 30% of total runtime while inter-node InfiniBand accounts for over 60%, confirming that even with NCCL's optimizations, the network fabric remains the binding constraint.

The paper demonstrates that purpose-built GPU-native libraries can substantially outperform portable general-purpose libraries on the hardware they target. The use of NCCL rather than MPI is shown to deliver concrete benefits, and the slab-vs-pencil tradeoff is reopened as an active design question rather than a settled matter. The implementation is also notable for its careful attention to overlapping communication with computation, which the authors quantify as recovering roughly 15% of total runtime that would otherwise be lost to synchronous waits. The principal limitations are scope and portability. The library is tested only on NVIDIA hardware with NCCL and InfiniBand HDR, and the authors do not evaluate performance on AMD GPUs, Intel GPUs, or systems with different interconnect topologies. The slab decomposition's scaling ceiling, while not encountered in the tested configurations, places a hard limit on the library's applicability to future systems with much larger GPU counts. The implementation is also

presented as research code rather than a production library, and the paper does not document an API or installation procedure of the kind that would support broad adoption.

2.7 Li et al. (2021): tcFFT — A Fast Half-Precision FFT Library for NVIDIA Tensor Cores [7]

Li and colleagues' CLUSTER 2021 paper introduces tcFFT, an FFT library that maps butterfly computations onto NVIDIA's Tensor Cores. This specialized matrix-multiplication units originally introduced in the Volta architecture (2017) to accelerate deep-learning workloads. On modern GPUs such as the V100 and A100, Tensor Cores deliver several times the floating-point throughput of the standard CUDA cores that conventional FFT libraries use. The V100 provides 125 teraFLOP/s of half-precision Tensor Core throughput compared to roughly 14 teraFLOP/s of single-precision CUDA core throughput, while the A100 widens this gap further. If FFT computations can be reformulated to exploit Tensor Cores, an order-of-magnitude increase in arithmetic throughput becomes available. The paper investigates whether this potential can be realized in practice, given that Tensor Cores are designed for dense matrix multiplication rather than the structured sparse computations of the FFT.

The library's central technical contribution is a reformulation of the radix-N butterfly stage as a small dense matrix multiplication. The authors observe that an N-point DFT can be expressed as a matrix-vector product where the matrix is the DFT matrix W with entries W[i,j] = ω^(ij), and that a batch of B such DFTs can be expressed as a matrix-matrix product between W and a B-column matrix of inputs. By choosing N to match the dimensions of a Tensor Core's hardware-supported matrix shapes (typically 16 by 16 for half-precision operations), the authors map small FFT stages directly onto Tensor Core fragment operations. Larger FFTs are constructed by composing these Tensor-Core-accelerated small stages using the standard Cooley–Tukey decomposition, with twiddle factor multiplications applied between stages. The implementation includes additional optimizations targeted at the GPU memory hierarchy: data is staged through shared memory in patterns chosen to maximize Tensor Core throughput, and the authors implement a custom data-layout transformation that reorders inputs to match the access pattern Tensor Cores require. All computation is performed in half precision (FP16) for inputs and accumulation in single precision (FP32) where the Tensor Core hardware supports it.

The paper reports speedups of up to 3.24 times over cuFFT on one-dimensional batched FFTs and up to 1.93 times on two-dimensional FFTs across a range of transform sizes on V100 and A100 GPUs. For very small transforms (sizes below 64), the speedup is less pronounced because the Tensor Core's fixed matrix shapes do not amortize their setup cost effectively. For very large transforms, the speedup also diminishes because memory bandwidth becomes the binding constraint, confirming once again that even Tensor-Core-accelerated FFTs ultimately run into the bandwidth wall that has shaped the entire field. The authors quantify the precision cost of moving from single to half precision: relative error on the order of 10^{-3} for typical inputs, compared to 10^{-7} for single precision cuFFT. They argue that this precision is acceptable for many scientific and machine-learning applications, particularly those involving frequency-domain convolutions in neural networks, where downstream operations tolerate or even benefit from reduced precision.

The principal strength of tcFFT is that it demonstrates a previously unexploited path to GPU FFT acceleration that does not depend on improvements to the memory hierarchy or interconnect. By exploiting hardware that was already present on the GPU but not previously used for FFT computation, the library achieves speedups that compound with rather than substitute for the optimizations of conventional libraries. The matrix-multiplication formulation is also elegant from an algorithmic standpoint, treating the DFT matrix as a first-class object rather than implicitly through butterfly diagrams. The principal limitation is the precision tradeoff. Half-precision FFTs are not appropriate for many scientific applications, particularly those involving long iterative simulations where round-off errors accumulate, and the authors do not extend their approach to single or double precision in this paper. A second limitation is that the library is single-GPU and does not address multi-node scaling, leaving open the question of whether Tensor Core acceleration can be combined with distributed-memory pencil or slab decompositions to compound their respective benefits. The library also targets only NVIDIA Tensor Cores and does not generalize to similar matrix units on AMD or Intel hardware.

2.8 Yu et al. (2023): MFFT — A GPU-Accelerated Mixed-Precision Large-Scale FFT Framework [8]

The MFFT paper presents a distributed FFT framework whose central innovation lies in how data is represented during inter-node communication. The authors observe that on large GPU-equipped systems, the all-to-all transpose operation that distributed 3D FFTs require has come to dominate runtime, and that this operation transfers data whose precision exceeds what is necessary for the algorithm's overall numerical accuracy. They propose that the data being communicated can be compressed into a reduced-precision format during transfer and then expanded back to full precision for local computation, thereby reducing the volume of data on the wire without compromising the precision of the arithmetic itself. The paper develops this idea into a complete framework, including a custom number format, GPU-resident compression and decompression kernels, and a full distributed FFT implementation built on top of these components.

The technical core of MFFT is a shared-exponent floating-point format the authors call sef32. In the standard IEEE 754 single-precision format, each 32-bit number carries its own 8-bit exponent and 23-bit mantissa. The authors observe that when a block of 32 floating-point values is transferred together, those values often have similar magnitudes, meaning their exponents are highly correlated. The sef32 format exploits this correlation by storing a single shared exponent for a block of values along with reduced-width mantissas, achieving an effective per-value width of approximately 16 bits while preserving most of the precision of the original 32-bit representation. Compression and decompression are implemented as GPU kernels that fuse with the existing data-packing and unpacking operations of the all-to-all transpose, so the precision conversion adds little overhead beyond what the transpose was already performing. The authors also implement an adaptive scheme in which the compression ratio can be selected based on the dynamic range of the data being transferred, with full single-precision communication retained

as a fallback for blocks whose values span too wide a range to share a single exponent without unacceptable precision loss.

The framework is evaluated on a system with up to 4,096 NVIDIA GPUs running 3D FFTs of size up to 8192^3. On the largest configuration, MFFT achieves parallel efficiency of 83.8%, compared to 53.2% for the CPU-based reference library used as a baseline. The reported end-to-end speedup at 4,096 GPUs is approximately 9.6 times over the baseline, with the compression of communicated data accounting for the largest single contribution. The authors also evaluate accuracy: relative error introduced by sef32 compression is reported as approximately 10^{-5} for typical scientific inputs, which is intermediate between half precision (10^{-3}) and single precision (10^{-7}). They argue that this accuracy level is sufficient for most FFT-using scientific applications, including molecular dynamics, turbulence simulation, and large-scale image processing pipelines, while noting that applications requiring full single-precision accuracy throughout can disable the compression at the cost of some performance. Profiling data show that on the 4,096-GPU configuration, communication time drops from approximately 78% of total runtime in the uncompressed baseline to approximately 52% with sef32 compression, redistributing the runtime profile toward a less communication-dominated regime.

The paper shows the advantage of separating communication precision from computation precision. By allowing the two to be set independently, MFFT preserves the numerical behavior of single-precision FFT computation while paying only a reduced-precision communication cost. The shared-exponent format is also clever as a practical engineering choice: it requires no hardware support beyond standard GPU integer and floating-point operations, and the compression and decompression kernels are inexpensive enough to be absorbed into existing data-packing operations without restructuring the surrounding library. The principal limitations are the dependence on the data's dynamic-range characteristics and the lack of formal error analysis. The accuracy guarantees offered by sef32 are empirical rather than theoretical, and the authors do not characterize how compression error compounds across iterative applications such as time-stepped simulations where the FFT is invoked thousands of times in sequence. A second limitation is that the framework is implemented and evaluated only on NVIDIA hardware; whether the approach generalizes to systems with different memory bandwidth, interconnect topology, or arithmetic characteristics is not established within the paper. Finally, the framework requires that applications opt into the reduced-precision communication, meaning existing codes must be modified to use it rather than benefiting transparently.

2.9 TurboFFT (2024): A High-Performance Fast Fourier Transform with Fault Tolerance on GPU [9]

TurboFFT paper addresses a failure class that is important as HPC systems grow in size and complexity. Silent data corruption (SDC), in which a hardware fault causes a computation to produce an incorrect result without any indication that an error has occurred. On systems with thousands of GPUs running for hours or days, the probability that at least one bit-flip will occur

during a single computation becomes non-negligible, and for FFT-based simulations such errors can corrupt downstream scientific results in ways that are difficult to detect after the fact. The authors propose that fault tolerance should be integrated directly into the FFT kernel itself rather than handled by external mechanisms such as checkpoint-restart, and they present TurboFFT as a GPU FFT library that combines competitive performance with on-line error detection and correction.

The fault-tolerance mechanism is based on algorithm-based fault tolerance (ABFT), a technique in which mathematical invariants of the algorithm are checked at runtime to detect errors. For the FFT specifically, the authors exploit a property of the DFT known as Parseval's relation, which states that the sum of squared magnitudes of a signal in the time domain equals the sum of squared magnitudes of its frequency-domain representation, up to a normalization factor. This relation provides a checksum: if the input and output sums do not match within a tolerance, an error has occurred somewhere during the computation. The authors extend this basic idea by computing weighted checksums that allow not only detection but also localization of errors to specific stages of the FFT, enabling targeted recomputation of the affected stage rather than restart from the beginning. Checksum computation is fused with existing FFT kernels so that the additional arithmetic operations execute alongside the productive computation rather than as separate passes over the data. The library is built as a co-designed system in which the FFT algorithm, the checksum scheme, and the GPU kernel implementation are developed together rather than treated as separable layers.

Performance is evaluated on NVIDIA A100 and H100 GPUs across a range of one-, two-, and three-dimensional transforms in single and double precision. The authors report that TurboFFT achieves performance competitive with cuFFT on most transform sizes, with a measured fault-tolerance overhead of 7% to 15% - substantially lower than the overhead of generic ABFT approaches applied to existing FFT libraries, which typically incur 30% or more. For some transform sizes, TurboFFT actually outperforms cuFFT despite providing fault tolerance, which the authors attribute to optimizations they introduced in the underlying FFT implementation that are independent of the fault-tolerance machinery. The paper includes fault-injection experiments in which random bit-flips are deliberately introduced into the computation: TurboFFT detects 100% of injected errors that affect the output and successfully recovers from over 99% of them, with recovery time dominated by the recomputation of the affected stage rather than by the detection itself. The authors also demonstrate that the checksum-based detection has a false-positive rate below 0.1% under normal operation, meaning that the overhead of unnecessary recomputation in the fault-free case is negligible.

The paper's principal strength is the integration of fault tolerance directly into a high-performance kernel without sacrificing competitive performance. ABFT for FFT has been studied previously, but typical implementations have treated the checksum as an external wrapper around an unmodified FFT, paying the full overhead of additional kernel launches and memory traffic. By co-designing the FFT and the checksum, the authors absorb most of the

additional cost into operations that the GPU was already performing, achieving overhead numbers that are low enough to make routine deployment plausible rather than reserved for situations where fault tolerance is explicitly required. The localization capability (the ability to identify which stage of the FFT contained the error) is also a meaningful advance over prior ABFT-FFT work, which typically provided only detection. The principal limitations involve the scope of faults the system can address. The Parseval-based checksum detects errors that change the numerical content of the computation but cannot detect errors that affect control flow, such as a corrupted loop counter that causes a stage to be skipped entirely. The authors acknowledge this and note that complementary techniques would be required for full coverage. The library is also single-GPU and does not address fault tolerance at the multi-node level, where additional failure modes such as interconnect errors and node failures arise. Finally, the recovery mechanism's effectiveness depends on the fault rate being low enough that single-stage recomputation is generally sufficient; under high fault rates, the cumulative overhead would grow significantly.

TurboFFT establishes that fault tolerance need not be a substantial performance tax for FFT computations on modern GPUs, provided the detection and recovery machinery is designed jointly with the FFT kernel itself. The work makes a concrete software contribution and demonstrates empirically that ABFT can be implemented at low overhead on current accelerator hardware.

2.10 M³XU (2024): Achieving High-Precision and Complex Matrix Multiplication on Tensor Core Architectures [10]

Tensor cores and similar matrix engines deliver enormous arithmetic throughput, but they natively support only reduced precision formats and only real-valued operands. FFT computations require complex arithmetic, and many scientific applications require single or double precision rather than half precision. The authors propose M³XU, a software framework that constructs high precision and complex valued matrix multiplications from sequences of low precision real valued matrix multiplications executed on existing tensor core hardware. The work is positioned as enabling FFT and other complex valued numerical kernels to fully exploit tensor core throughput without requiring hardware changes and without accepting the precision compromises that earlier tensor core FFT implementations have made.

M³XU uses a decomposition strategy for high precision arithmetic. To compute a single precision matrix product on a tensor core that natively supports only half precision multiplication, the authors split each single precision input into a sum of two half precision components (a high part and a low part) and express the single precision product as a sum of four half precision products that can be computed independently on the tensor core hardware. The four partial products are then accumulated in single precision to recover the full result. A similar but more complex decomposition is used for double precision computation built from single precision tensor core operations. For complex valued multiplication, the authors decompose each complex matrix product into four real matrix products corresponding to the real and imaginary parts of the inputs, again executed independently on the tensor core hardware and combined afterward. The framework includes optimizations that exploit the structure of these decompositions to reduce redundant work, in particular fusing the partial product accumulations with the tensor core fragment loads to avoid extra round trips through the GPU memory hierarchy. The implementation is provided as a library that can be integrated into existing GPU codes through a familiar matrix multiplication API.

The framework is evaluated on NVIDIA H100 GPUs across a range of synthetic matrix multiplication benchmarks and several application kernels, including FFT. For single precision complex matrix multiplication, M³XU achieves throughput approximately 2.1 times higher than cuBLAS, the standard NVIDIA library for single precision matrix operations, while delivering single precision accuracy verified against a double precision reference. For double precision complex matrix multiplication, the speedup over cuBLAS is approximately 1.4 times, smaller than the single precision case because the four-way decomposition required to build double precision from single precision is more expensive than the corresponding decomposition for single precision from half precision. When applied as the matrix multiplication backend for an FFT implementation similar in structure to tcFFT but operating in single precision, M³XU enables FFT throughput approximately 1.7 times that of cuFFT on the same hardware. Accuracy

measurements confirm that the framework delivers numerical results equivalent to standard single precision or double precision computation, with relative errors at the expected level for the target precision.

It establishes a general method for using tensor core hardware in contexts the hardware was not designed for. The decomposition strategy is mathematically rigorous in the sense that the partial products combine to produce the exact correctly rounded result of the higher precision operation, up to the floating point rounding behavior of the accumulator. This makes M³XU suitable for applications that cannot tolerate the precision loss of native tensor core operation but want to benefit from the hardware's throughput. The complex valued support is also a meaningful contribution, since it brings tensor cores into reach for the broad class of numerical kernels that operate on complex data. The principal limitations involve the cost of the decomposition itself. Building a higher precision operation from multiple lower precision operations requires several times more underlying tensor core invocations than a native operation would, so the speedups achieved depend on the ratio between tensor core throughput and the throughput of the standard arithmetic units that would otherwise be used. On hardware where this ratio is small, the benefits of M³XU diminish or disappear. The framework is also implemented only for NVIDIA tensor cores and does not address similar matrix engines on AMD or Intel hardware, although the underlying decomposition strategy is hardware agnostic in principle. Finally, the FFT evaluation is single GPU, leaving open questions about how the matrix multiplication overhead structure interacts with distributed memory communication at scale.

2.11 Servodio and Li (2024): Memory Efficiency Oriented Fine-Grain Representation and Optimization of FFT [11]

Servodio and Li's MEMSYS 2024 paper revisits the FFT performance question from a perspective that has received less attention in the recent HPC literature: how the algorithm's interaction with the memory subsystem can be optimized at a fine granularity below the level addressed by typical library design. The authors observe that contemporary FFT libraries treat memory as a collection of bandwidth and capacity numbers, but that on modern HPC systems the memory subsystem has significant internal structure including memory channels, cache hierarchies, and increasingly heterogeneous mixes of DRAM, high-bandwidth memory (HBM), and persistent memory. They argue that the FFT's data access pattern can be characterized at a much finer grain than current libraries do, and that this characterization can be used to drive optimizations such as data placement, prefetching, and access scheduling that yield measurable performance improvements over libraries that treat memory as a black box.

The paper develops a representation framework in which an FFT computation is decomposed into a sequence of memory access events, each tagged with attributes including the address range touched, the access width, the temporal ordering relative to other events, and the dependency relationship with subsequent computation. This representation is constructed automatically from a high-level FFT specification, allowing the framework to analyze and optimize memory behavior without requiring the user to write low-level code. The framework includes a set of optimization passes that operate on this representation. One pass identifies opportunities for software prefetching by analyzing the spacing between memory accesses and inserting prefetch instructions sufficiently far in advance to mask memory latency. Another pass groups accesses that touch the same cache line into clusters that can be performed together, reducing the number of cache misses. A third pass partitions the data layout across memory channels to balance bandwidth utilization, exploiting the fact that interleaved access patterns can saturate channel bandwidth more effectively than sequential ones for certain transform sizes. The framework targets CPU-based execution rather than GPUs, focusing on systems where the memory hierarchy has many tunable layers and where library-level optimizations such as those in FFTW have left fine-grain memory behavior under-exploited.

The authors evaluate their framework on a multi-socket Intel Xeon system with HBM and DDR5 memory, compared against FFTW and Intel MKL. For one-dimensional FFTs in the size range relevant to scientific applications, the framework achieves performance improvements of approximately 15% to 30% over FFTW and 5% to 20% over MKL across most transform sizes. The improvements are larger for transformers whose working sets exceed cache capacity, where the memory access pattern dominates runtime, and smaller for transforms that fit entirely in cache, where arithmetic throughput is the limiting factor. The authors also report results for two-dimensional and three-dimensional transforms, where the speedups are somewhat smaller

because the existing libraries already include optimizations targeted at the higher-dimensional access patterns. A detailed breakdown of the contribution from each optimization pass shows that prefetching and channel partitioning together account for the majority of the gain, while access clustering provides smaller but consistent improvements.

The paper poses the memory subsystem as an active design dimension for FFT optimization rather than treating it as a fixed background against which arithmetic is scheduled. By representing memory accesses explicitly and applying targeted optimization passes, the framework captures performance opportunities that library-level autotuning misses because the autotuner cannot see below its own abstraction layer. The principal limitations relate to scope and portability. The framework is implemented for CPU systems with conventional memory hierarchies and does not address GPUs, where the memory model is sufficiently different that the optimizations would need to be redesigned. The performance improvements, while consistent, are modest compared with the order-of-magnitude gains reported by GPU and Tensor Core papers, reflecting the more constrained design space available on CPUs at this point in the HPC technology timeline. The framework also requires that the FFT be expressed in the authors' representation, which is automated for standard transform shapes but would require manual effort to extend to non-standard variants such as pruned or sparse FFTs.

2.12 Aseeri (2025): Distributed Memory Fast Fourier Transforms in the Exascale Era [12]

The paper organizes the distributed FFT design space along several axes. The first is decomposition strategy, where the author surveys the contrast between slab and pencil decompositions and notes that the right choice depends on the ratio of processors to grid size, the interconnect topology, and whether the transforms are being performed on CPUs or GPUs. The second axis is communication library, where Aseeri compares MPI-based implementations with NCCL-based implementations and notes that the gap between them has narrowed as MPI implementations have added GPU-aware features but that NCCL still retains a meaningful advantage on dense GPU systems with NVLink. The third axis is precision and numerical behavior, where the author surveys the recent work on mixed-precision and reduced-precision FFTs and discusses the tension between performance gains and accuracy guarantees. The fourth axis is software engineering quality, including portability across architectures, ease of integration with existing applications, and the maintenance burden imposed by libraries that depend tightly on specific hardware features. The paper presents performance comparisons drawn from the author's own benchmarking work and from the published results of recent libraries including heFFTe, cuFFTMp, and the multi-GPU implementations developed at NVIDIA.

The paper observes the state of the field rather than specific quantitative results. Aseeri notes that the order-of-magnitude performance improvements that characterized the GPU-acceleration era have plateaued, with recent libraries achieving incremental rather than transformative speedups over their predecessors. The communication bottleneck that has

dominated distributed FFT performance for over a decade remains the constraint, and recent improvements such as NCCL adoption and reduced-precision communication have slightly improved but have not eliminated it. The author identifies several open problems that the field continues to grapple with, including how to fully exploit hierarchical interconnect topologies in which the bandwidth ratio between intra-node and inter-node communication can exceed an order of magnitude, how to integrate fault tolerance into production libraries without prohibitive overhead, how to support precision flexibility in a way that applications can actually use, and how to manage the software complexity that comes with supporting multiple GPU vendors and interconnect technologies. The paper also discusses the challenge of benchmarking, noting that fair comparisons between libraries are difficult because different libraries make different assumptions about input sizes, data layouts, and the cost of one-time setup operations that may or may not be amortized over many transforms.

This paper focuses on perspective. By stepping back from any single library or technique, Aseeri provides a map of the field that is useful both for identifying which contributions matter relative to each other and for orienting future research efforts toward the problems that remain unsolved. The paper draws on knowledge of FFT benchmarking and references work from research groups around the world, giving the survey a wholistic perspective. The paper also reflects the perspective of a single author and the research communities she works closely with, which inevitably shapes which contributions receive the most attention.

3 Comparative Discussion

The twelve papers summarized in Section 2 span twenty years of research, from FFTW3 in 2005 [1] to Aseeri's survey in early 2025 [12]. Read in chronological order, the lineup still uses the same underlying algorithm, the Cooley-Tukey FFT, has been repeatedly redesigned to fit the changing hardware on which it runs. CPU autotuning gave way to GPU acceleration, single-GPU optimization gave way to multi-node distribution, and floating-point throughput gave way to communication bandwidth as the binding constraint. This paper will identify 6 main topics to the papers.

3.1 Communication Cost

The clearest pattern across the twelve papers in this review is that communication cost has been the binding constraint on distributed FFT performance for over a decade, and that this remains true on the most recent hardware. The pattern is established theoretically in Czechowski et al. [3] and confirmed empirically by every distributed library that follows. Czechowski's lower-bound analysis [3], derived from communication-avoiding algorithms theory, predicts that on exascale-class systems distributed 3D FFTs will spend 90% or more of their runtime in interconnect communication. When heFFTe ran on Summit eight years later [5], the measured figure was 97% on the GPU configuration and roughly 50% on the CPU configuration. On CPUs, local FFT computation is slow enough that arithmetic and communication consume roughly equal shares of runtime [5]. On GPUs, local FFTs accelerate by more than 40 times relative to CPU baselines [5], which leaves the interconnect as nearly the only thing that takes any meaningful time at all. GPU acceleration does not solve the communication problem, but rather it intensifies it.The papers that follow heFFTe can be read as different responses to this same pressure. Verma et al. [6] attempt to attack the communication side directly by switching from MPI to NCCL and by reducing the number of all-to-all transposes from two (in pencil decomposition) to one (in slab). Their result of approximately 4× speedup over heFFTe at moderate GPU counts [6] shows that the choice of communication library and decomposition strategy is not settled simply because heFFTe exists. MFFT [8] takes a different approach to the same problem, leaving the decomposition unchanged but compressing the data being transferred so that less bandwidth is consumed per FFT. The reported reduction in communication time from 78% to 52% of total runtime on a 4,096-GPU configuration [8] confirms that the bandwidth wall can be partially scaled by reducing demand rather than increasing supply. Both approaches are partial. Verma's slab decomposition reaches its scalability ceiling at large enough processor counts [3][6], and MFFT's compression depends on the dynamic range of the data being transferred [8]. The bottleneck has been moved but not removed. Aseeri's 2025 synthesis [12] confirms what the trend line already suggests, which is that the order-of-magnitude performance improvements of the early GPU era have given way to incremental gains, and that the remaining gains are increasingly difficult to extract because they require working against the interconnect

rather than around it. From these papers, it can be concluded that no single intervention solves the communication problem, and that progress comes from layering partial improvements at different levels of the stack.

3.2 The Slab Versus Pencil Decomposition Debate

Another central debate concerns how the 3D data grid should be partitioned across processors. The two original choices, slab decomposition and pencil decomposition, were introduced in Section 1, but their relative merits have been debated across the papers. This can be correlated to hardware evolution. As hardware improves, it becomes much easier to perform certain calculations, reviving design questions that the field had considered settled.

Czechowski et al. provide the theoretical framing. Slab decomposition requires only one all-to-all transpose per FFT and is therefore lower in communication volume, but it limits scalability to $P \leq N$ processors for an N^3 grid. Pencil decomposition raises the scalability ceiling to $P \leq N^2$ at the cost of requiring two transposes. Czechowski et al. show that the asymptotic gap between the two narrows at large processor counts, but that for any finite system the choice depends on the specific values of P and N and on the relative cost of the two transposes versus a single transpose with more data per transfer.

P3DFFT in 2012 commits to pencil decomposition and demonstrates that the choice scales: 45% weak-scaling efficiency from 128 to 65,536 cores on a Cray XT5. At this scale the pencil approach is unambiguously correct, because the slab approach's $P \leq N$ ceiling would have been hit by a factor of eight or more. Pekurovsky's library establishes pencil decomposition as the default for HPC FFT libraries, and heFFTe in 2020 inherits this default, although it does provide a slab mode for cases in which the processor count is small enough to make the slab approach favorable.

Verma et al. in 2022 reopen the debate. Their argument is that on dense GPU systems with eight A100 GPUs per node, the number of processors involved in any realistic computation is far smaller than the linear grid dimension. A 4096^3 grid distributed across 256 GPUs has $P \approx 256$ and $N = 4096$, so the slab approach's $P \leq N$ ceiling is far above the actual processor count. Under these conditions slab decomposition is not only viable but preferable, because it requires only one transpose instead of two. Verma et al.'s measurement of approximately 4× speedup over heFFTe on equivalent hardware quantifies the cost of using the wrong decomposition for the system: a library defaulted to pencil decomposition pays roughly twice the communication cost it needs to, and on systems where communication is 60-90% of total runtime, that doubling is significant.

Aseeri's 2025 survey concludes that the right choice depends on the ratio of processors to grid size, the interconnect topology, and whether the transforms are being performed on CPUs or GPUs. This is a pragmatic answer rather than a theoretical resolution. The slab-versus-pencil

question does not have a single correct answer; it has different correct answers for different hardware regimes. CPU-based petascale systems with hundreds of thousands of cores justify pencil decomposition. GPU-based systems with hundreds of accelerators justify slab decomposition. Future systems with tens of thousands of GPUs may swing the answer back toward pencil decomposition or may motivate hybrid approaches.

What is striking about this debate is how directly it illustrates the dependence of algorithmic choices on hardware parameters. The slab decomposition was widely regarded as a legacy approach for most of the 2010s, displaced by pencil decomposition for serious HPC use. Verma et al. did not introduce a new technique; they revived an old one and showed that hardware evolution had made it relevant again. This kind of revival is rare in HPC research and worth noting. It suggests that even algorithmic ideas that appear to have been superseded should not be discarded, because changes in the underlying hardware can return them to relevance.

3.3 Altering Precision as a Design Choice

The third theme concerns numerical precision, which several papers in the lineup treat as a design dimension that can be varied to trade accuracy for performance (as opposed to a fixed parameter). The papers show the field gradually learning that precision is not a single choice but several independent design choices, each of which can be set differently.

Frigo and Johnson's FFTW3 supports both single and double precision but treats them as alternative configurations of the same library; the user picks one and the entire computation runs at that precision. This is the conventional view of precision as a global property of the FFT.

tcFFT in 2021 introduces the first significant departure. By targeting Tensor Cores, which natively support only half-precision arithmetic with single-precision accumulation, the library shifts the entire FFT computation to FP16 and accepts a precision cost of roughly 10^{-3} relative error compared to single precision's 10^{-7}. The authors justify this tradeoff by arguing that many applications, particularly machine-learning convolutions, can tolerate the precision loss in exchange for the throughput gain. The contribution here is not novel hardware exploitation alone but the framing of precision as something an application can choose.

MFFT in 2023 introduces the more significant conceptual move. The authors observe that computation precision and communication precision can be set independently. The arithmetic in MFFT is performed at full single precision, but the data transferred between nodes during the all-to-all transpose is compressed to a shared-exponent format with effective per-value width near 16 bits. The result is that the numerical behavior of the FFT remains close to single precision, while the communication cost is closer to half precision. This decoupling of precision from communication is one of the more elegant ideas in the entire lineup, because it captures most of the performance benefit of reduced precision without paying the full accuracy cost.

M³XU in 2024 represents the inverse move. Rather than accept the precision loss of native Tensor Core operation, the authors construct single-precision and double-precision matrix

multiplications from sequences of half-precision Tensor Core operations. The partial products combine to produce the correctly rounded high-precision result, up to the floating-point rounding behavior of the accumulator. This recovers the precision that tcFFT sacrificed while still exploiting the Tensor Core hardware that motivated tcFFT in the first place. The cost is that building a higher-precision operation from multiple lower-precision operations requires several times more underlying Tensor Core invocations than a native operation would, so the effective speedup over standard cuBLAS is smaller than tcFFT's speedup over cuFFT.

TurboFFT in 2024 connects precision to fault tolerance. The Parseval-relation checksum that the library uses for error detection is itself a numerical computation, and its sensitivity to silent data corruption depends on the precision at which it is performed. The authors do not make precision a primary subject of their paper, but their decision to support both single-precision and double-precision FFT computation, with checksum overhead reported separately for each, illustrates that precision interacts with fault tolerance in ways that single-precision-only or double-precision-only libraries do not need to consider.

Aseeri's 2025 survey identifies precision flexibility as one of the open problems the field continues to grapple with. The author notes a tension between performance gains and accuracy guarantees, and observes that supporting precision flexibility "in a way that applications can actually use" is more difficult than supporting it as a configuration option that requires the application developer to manually choose. This is a software engineering concern as much as a numerical one, and it reflects the gap between research libraries that demonstrate what precision flexibility can deliver and production libraries that have to expose it through usable APIs.

In 2005 precision was a global compile-time choice. By 2025 it had been split into computation precision, communication precision, accumulation precision, and checksum precision, each of which can be set independently in principle. Whether applications and library APIs have caught up with this disaggregation is a separate question, and one to which the field's answer is still incomplete.

3.4 Bottleneck after Bottleneck

A fourth pattern is that each generation of FFT research finds a new bottleneck after the previous one has been partially addressed.

Frigo and Johnson [11] in 2005 located the binding constraint at the CPU cache hierarchy. Their autotuning approach is fundamentally about adapting to the cache sizes and SIMD widths of different processor architectures, because at the time the bottleneck for a single-node FFT was how efficiently the algorithm could be staged through L1, L2, and L3 caches. The genfft codelet generator and the runtime planner together address this constraint by selecting the algorithmic decomposition that best matches the target machine's memory hierarchy. This is the right place to focus optimization effort in 2005, when single-node performance is the relevant question.

Govindaraju et al. [6] three years later relocated the binding constraint to GPU shared memory and global memory bandwidth. The authors observe that on the 8800 GTX, global memory accesses are far more expensive than arithmetic, so the FFT's stride-doubling memory access pattern becomes the bottleneck rather than the arithmetic itself. Their hierarchical FFT, which performs small stage groups within shared memory before writing intermediate results back to global memory, is fundamentally a response to this new constraint. The optimization techniques are different from FFTW3's, but the underlying logic is the same: identify the layer of the memory hierarchy where time is being spent and target that layer specifically.

Ayala et al. [4] in 2020 push the binding constraint further out, from on-chip and on-node memory to inter-node interconnect. heFFTe's measurement that 97% of GPU FFT runtime is consumed by MPI communication is the empirical confirmation that the optimization frontier has moved from the memory hierarchy to the network fabric. Their library's contributions, custom data-packing kernels, GPU-aware MPI integration, autotuned communication patterns, are all aimed at this new frontier. The local FFT computation, which earlier papers had labored to optimize, is now small enough that further work on it would yield negligible gains.

MFFT [14] continues this migration by attacking communication volume rather than communication mechanism. By compressing the data transferred during the all-to-all transpose, the framework reduces the time spent on the wire without changing the underlying communication library or topology. This is a logical next step once the communication mechanism itself has been optimized: if the network is already being used as efficiently as possible, the only remaining lever is to send less data through it.

Servodio and Li [1] return to a layer that the field had largely set aside. After more than a decade of GPU and interconnect optimization, their MEMSYS 2024 paper revisits the CPU memory subsystem and finds 15-30% performance gains over FFTW that library-level autotuning had missed. This is initially surprising, given that FFTW was specifically designed to optimize for the memory hierarchy. The explanation is that FFTW's autotuning operates above a level of abstraction that hides certain details, including memory channel partitioning, software prefetch scheduling, and cache-line clustering. By representing memory access events explicitly and applying optimization passes at this finer grain, Servodio and Li find headroom that the higher-level autotuner could not see. The locus of optimization has not only moved over time but has periodically returned to layers the field thought were exhausted.

TurboFFT [13] represents a different kind of relocation: not to a new layer of the system, but to a new dimension of the problem. Where previous papers all aim at runtime performance, TurboFFT introduces fault tolerance as a co-equal optimization target. The 7-15% overhead the library accepts is a deliberate cost, paid in exchange for detection and recovery of silent data corruption. This is the first paper in the lineup to argue that performance is not the only thing worth optimizing for; correctness, in the presence of hardware faults, is itself a performance

concern when the alternative is a corrupted scientific result that must be recomputed from scratch.

What this theme illustrates is that HPC research is not a steady accumulation of improvements at every layer but a sequence of focused efforts that each address the current binding constraint. The field's sense of which constraint is binding has shifted multiple times in the period covered by the lineup, and the shifts have generally been driven by hardware changes: GPUs introduced bandwidth-constrained execution, exascale interconnects introduced communication-constrained execution, and accelerator scale introduced fault-tolerance concerns. Aseeri's 2025 survey [12] suggests that the next constraint may be heterogeneity itself, the difficulty of writing code that performs well across NVIDIA, AMD, and Intel GPUs simultaneously, but this is a constraint the field has only begun to grapple with.

3.5 Production Versus Research

The gap between research prototypes and production libraries are also a commonality many papers that develop their own libraries. Several of the papers surveyed are research demonstrations that show what is possible; others are software contributions that real applications can adopt. The two are not equally useful.

The clearest examples of production readiness in the lineup are P3DFFT [4] and heFFTe [4]. P3DFFT was released as open-source software with documentation, sample programs, a stable API, support for multiple compilers and platforms, and integration paths into existing scientific applications. The library has been used in turbulence simulations, climate modeling, astrophysics, and material-science codes. This adoption is itself empirical validation that the library's performance claims hold under real working conditions. heFFTe carries similar production-readiness commitments and additionally provides Spack-based installation, validation suites, and drop-in replacements for legacy library APIs. The 1.5x end-to-end application speedup heFFTe demonstrates on the LAMMPS Rhodopsin protein benchmark [4] is meaningful in a way that synthetic FFT benchmarks are not, because it measures the performance gain that real users will experience.

Several other papers in the lineup are explicitly research demonstrations rather than production libraries. Verma et al. [3] note that their implementation is research code rather than a production library, and the paper does not document an API or installation procedure. MFFT [14] is implemented and evaluated only on NVIDIA hardware, and its use requires applications to opt into reduced-precision communication. TurboFFT [13] reports impressive fault-tolerance results, but the library's deployment requirements, particularly its single-GPU scope, mean it cannot be used directly by the multi-node applications that most need fault tolerance. M³XU [12] provides a library API but is single-GPU and has not been integrated into any major distributed FFT framework.

Demonstration libraries serve the legitimate purpose of establishing what is achievable, and many of the techniques they introduce are eventually adopted into production libraries. tcFFT's [12] Tensor Core formulation, for example, has influenced subsequent libraries that integrate Tensor Core acceleration into more general frameworks. The point is that the performance numbers reported by research libraries should be understood as upper bounds on what the corresponding production library would achieve, and that the gap between demonstration and deployment can be substantial.

Aseeri [12] makes this concern explicit, identifying software engineering quality as one of the design axes along which distributed FFT libraries differ. Portability across architectures, ease of integration with existing applications, and the maintenance burden imposed by libraries that depend tightly on specific hardware features are all considered, and the survey notes that fair comparisons between libraries are difficult because different libraries make different assumptions about input sizes, data layouts, and the cost of one-time setup operations. This is a way of saying that benchmark numbers alone do not capture the practical usability of a library, and that production readiness is a dimension of evaluation distinct from raw performance.

A second observation under this theme is that production readiness does not correlate simply with performance. P3DFFT is highly production-ready but predates the GPU era and offers no GPU support. heFFTe is production-ready and supports GPUs but is slower than Verma et al.'s research library on the systems both target. The most production-ready library is not the fastest, and the fastest library is not the most production-ready. Application developers must choose between these dimensions rather than expecting both, and the choice depends on factors that the research literature does not always make visible.

4 Future Directives and Conclusion

This review traced the progression of high-performance computing techniques for the Fast Fourier Transform across two decades of research, from FFTW3's autotuned CPU implementation in 2005 to Aseeri's survey of the exascale era in 2025. The twelve papers examined demonstrate that the FFT has been repeatedly redesigned to fit the hardware on which it runs, with each generation finding new headroom in a different layer of the system: cache hierarchies in the CPU era, shared memory and global bandwidth in the single-GPU era, interconnect bandwidth and collective communication in the distributed era, and most recently specialized matrix units, mixed-precision communication, and fault-tolerance mechanisms in the exascale era.

Looking forward, several research directions emerge from the points identified across the twelve papers. A future direction could combine the specialization gains of Tensor Core libraries [12] and matrix-extension frameworks [10] with the distributed-memory frameworks of heFFTe [4] and its successors, since no current library uses both opportunities simultaneously. Another direction is the development of vendor-agnostic abstractions that allow specialization to be expressed portably.

Finally, the field could benefit from standardized benchmarking protocols that account for the production-readiness dimension Aseeri raises, so that fair comparisons between libraries reflect throughput, API stability, integration cost, and the assumptions each library makes about input sizes and data layouts. The most recent The next decade of HPC FFT research may be characterized by less individual breakthroughs and more by integration work that combines existing techniques.

5 Bibliography

[1] 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.

[2] N. K. Govindaraju, B. Lloyd, Y. Dotsenko, B. Smith, and J. Manferdelli, "High performance discrete Fourier transforms on graphics processors," in *Proc. 2008 ACM/IEEE Conf. Supercomputing (SC '08)*, Austin, TX, USA, Nov. 2008, pp. 1–12.

[3] K. Czechowski, C. Battaglino, C. McClanahan, K. Iyer, P. K. Yeung, and R. Vuduc, "On the communication complexity of 3D FFTs and its implications for exascale," in *Proc. 26th ACM Int. Conf. Supercomputing (ICS '12)*, Venice, Italy, Jun. 2012, pp. 205–214.

[4] D. Pekurovsky, "P3DFFT: A framework for parallel computations of Fourier transforms in three dimensions," *SIAM J. Sci. Comput.*, vol. 34, no. 4, pp. C192–C209, 2012.

[5] A. Ayala, S. Tomov, A. Haidar, and J. Dongarra, "heFFTe: Highly efficient FFT for exascale," in *Proc. Int. Conf. Computational Science (ICCS 2020)*, Lecture Notes in Computer Science, vol. 12137. Cham, Switzerland: Springer, 2020, pp. 262–275.

[6] M. K. Verma, R. Samtaney, and A. G. Chatterjee, "Scalable multi-node fast Fourier transform on GPUs," arXiv:2202.12756 [cs.DC], Feb. 2022.

[7] B. Li, S. Cheng, and J. Lin, "tcFFT: A fast half-precision FFT library for NVIDIA Tensor Cores," in *Proc. IEEE Int. Conf. Cluster Computing (CLUSTER 2021)*, Sep. 2021, pp. 1–11.

[8] Y. Hu, Y. Liu, X. Liao, S. Tu, J. Liu, and L. Xu, "MFFT: A GPU accelerated highly efficient mixed-precision large-scale FFT framework," *ACM Trans. Architecture and Code Optimization (TACO)*, vol. 20, no. 3, art. 43, pp. 1–23, Jul. 2023.

[9] S. Zhang, H. Lin, S. Di, J. Chen, Z. Chen, and F. Cappello, "TurboFFT: A high-performance fast Fourier transform with fault tolerance on GPU," arXiv:2405.02520 [cs.DC], May 2024.

[10] Y. Zhang, B. Yan, S. Yin, B. Sun, J. Yan, Q. Li, X. Liu, and Y. Wang, "M³XU: Achieving high-precision and complex matrix multiplication on Tensor Core architectures," in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC '24)*, Atlanta, GA, USA, Nov. 2024.

[11] S. Servodio and X. Li, "Memory efficiency oriented fine-grain representation and optimization of FFT," in *Proc. Int. Symp. Memory Systems (MEMSYS '24)*, Sep. 2024, pp. 245–256.

[12] S. A. Aseeri, "Distributed memory fast Fourier transforms in the exascale era," in *Proc. 2025 Int. Conf. Intelligent Control, Computing and Communications (IC3 2025)*, Mathura, India, Feb. 2025, pp. 1043–1046.

[13] 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, Apr. 1965.

[14] NVIDIA Corporation, "cuFFT library documentation," NVIDIA Developer. [Online]. Available: https://docs.nvidia.com/cuda/cufft/

[15] J. Heo, Y. Jung, S. Lee, and Y. Jung, "FPGA implementation of an efficient FFT processor for FMCW radar signal processing," *Sensors*, vol. 21, no. 19, art. 6443, Sep. 2021.

[16] M. R. Reddy, "FPGA implementation of pipeline digit-slicing multiplier-less radix-2 DIF SDF butterfly for Fourier transform structure," arXiv:1806.04570 [eess.SP], Jun. 2018.

[17] A. Ayala, S. Tomov, X. Luo, H. Shaiek, A. Haidar, G. Bosilca, and J. Dongarra, "Impacts of multi-GPU MPI collective communications on large FFT computation," in *Proc. IEEE/ACM Workshop on Exascale MPI (ExaMPI)*, Denver, CO, USA, Nov. 2019, pp. 12–18.

[18] U.S. Department of Energy, "Exascale Computing Project: Application development portfolio," Exascale Computing Project, Tech. Rep., 2021. [Online]. Available: https://www.exascaleproject.org/