connect·share·advance

PHUSE SDE Mumbai 2026

Performance & Efficiency Optimization in Modern R Clinical Programming

A Practical Guide to 10–100x Speed Improvements in Data Processing

Sivakumar G

Clinical Standards | Eli Lilly and Company

phuse.global

Agenda

connect·share·advance

Introduction & Motivation

Computational challenges in modern clinical trials

Results & Performance Gains

10–100x speedups demonstrated with benchmarks

Five Key Technologies

data.table, Arrow, DuckDB, dtplyr, furrr, bench

Validation & Best Practices

Regulatory compliance and adoption roadmap

Benchmark Methods

Realistic CDISC datasets & test scenarios

Conclusion & Recommendations

Strategic implementation guidance

Introduction — The Computational Challenge

connect·share·advance

The Problem

Modern clinical trials generate datasets with millions of records from real-time data sources (eCOA), wearable devices, PRO, and laboratory systems — creating significant bottlenecks in traditional R workflows.

- Extended processing times (hours to days)
- Memory overflow with large datasets
- No support for real-time data monitoring
- Delayed regulatory submissions
- High infrastructure compute costs

Adoption Barriers

Validation Concerns

Need for extensive testing and documentation in regulated environments

Knowledge Gaps

Limited awareness of modern optimization techniques among programmers

Legacy Codebases

Substantial investment in existing dplyr/tidyverse workflows

Risk Aversion

Conservative approach to adopting new technologies in pharma

Scope — Five Key Technologies

connect·share·advance

{data.table}

15–30×

faster

In-memory high-performance manipulation. 75% less memory than dplyr.

{arrow} & {duckdb}

100×

larger datasets

Out-of-memory columnar analytics. Handle datasets far beyond available RAM.

{dtplyr}

11×

speedup

Transparent optimization for dplyr with minimal code changes.

{furrr}

5×

parallel scaling

Near-linear parallel processing for TLF batch generation.

{bench}

µs

precision

Rigorous performance measurement and memory profiling.

Methods — Benchmark Design

connect·share·advance

Benchmark Dataset — Synthetic Phase 3 Oncology Trial

ADSL

500 records

Demographics & treatment assignments

ADAE

500K records

OCCDS structure, 30 AE terms

ADLB

5M records

30+ parameters × 25 visits/subject

Three Benchmark Scenarios

Complex AE Analysis

Subject-level TEAE summary with joins, aggregations and SOC-level incidence.

`data.table vs dplyr`

Out-of-Memory ADaM Derivation

Laboratory trend analysis where dataset exceeds available RAM.

`Arrow / DuckDB`

High-Volume TLF Generation

Produce 50 adverse event tables with different specifications in parallel.

`furrr (parallel)`

Results — Scenario 1: Complex AE Analysis | {data.table} vs {dplyr}

phuse

connect·share·advance

Task: Subject-level TEAE summary with joins, aggregations and SOC-level incidence rates (ADAE, 500,000 records).

Speedup Factor

15.9x
data.table Speedup

vs. traditional dplyr

Memory Savings

72%
Memory Reduction

Peak RAM savings

dtplyr Bridge

11x
dtplyr Speedup

Familiar dplyr syntax

Approach Comparison

Traditional dplyr
Execution: 45 min | Memory: High (3.2 GB) | Code Change: Baseline

`Baseline`

Optimized data.table
Execution: 2.5 min | Memory: Low (1.2 GB) | Code Change: Moderate rewrite

`15.9x faster`

dtplyr layer
Execution: 4 min | Memory: Low (2.3 GB) | Code Change: Minimal changes

`11x faster`

✦ Key Finding

{data.table} delivers 15.9x speedup with 72% memory reduction. {dtplyr} offers 11x improvement using familiar dplyr syntax with minimal code changes.

Results — Scenario 2: Out-of-Memory Analytics | {arrow} &{duckdb}

connect·share·advance

Task: Lab trend analysis for 5M records exceeding available RAM — traditional approach fails completely with out-of-memory error.

{arrow} / Parquet Approach

- ➔ Reads only required columns (columnar storage)
- ➔ 2–3x data compression vs CSV/SAS formats
- ➔ Zero-copy ops — eliminates data duplication
- ➔ 50–60% reduction in peak memory usage
- ➔ Supports datasets 10–100x larger than RAM
- ➔ Compatible with existing tidyverse code

⚡ 50–60% Memory Reduction • 10–100x Scale

{duckdb} — Embedded SQL Analytics

- ❖ Embedded analytical DB — no server required
- ❖ Familiar SQL syntax for SAS-to-R transition
- ❖ Vectorized execution for columnar data
- ❖ Seamless R integration via DBI interface
- ❖ Handles Parquet, CSV, and in-memory frames
- ❖ Ideal for complex multi-table JOINs at scale

⚡ Zero-Config SQL • Native R Integration

✦ Practical Note — Skipping the RAM Bottleneck

```
open_dataset("ADLB.sas7bdat", format="sas7bdat") → write_dataset(adlb_stream, "adlb_parquet/", format="parquet", partitioning="PARAMCD")
```

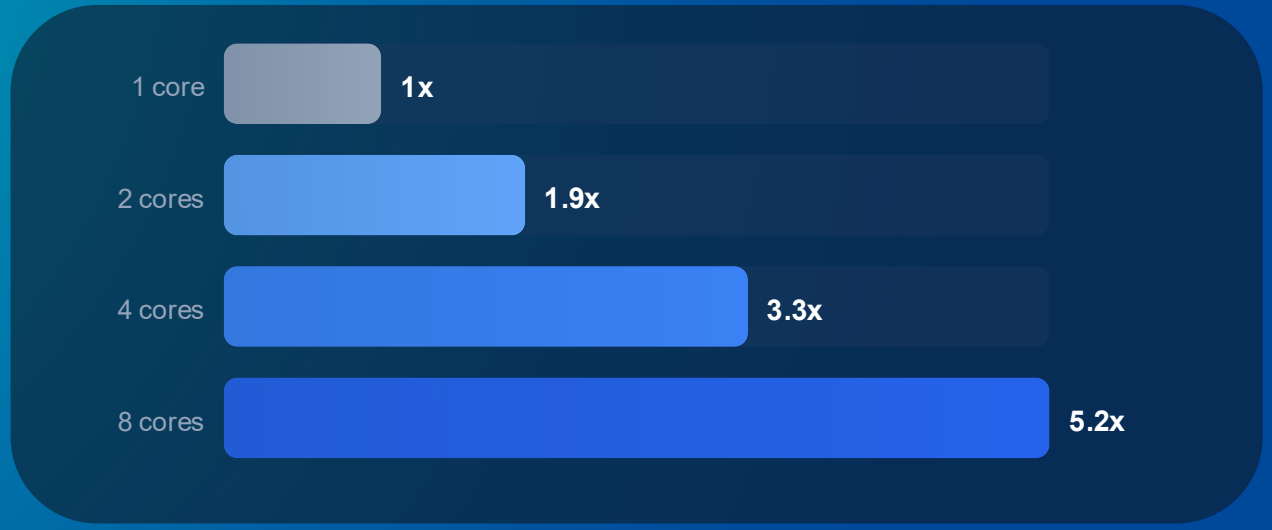
Results — Scenario 3: Parallel TLF Generation | {furrr}

connect·share·advance

Task: Generate 50 adverse event tables with different specifications. Sequential vs. parallel comparison.

Speedup	Core Efficiency	Time Saved	Optimal Cores
5.2x	83%	42→8s	4–6
with 8 cores	scaling at 4 cores	50 tables generated	beyond = diminishing returns

Scaling Efficiency by Core Count



Best Use Cases for {furrr}

- Batch TLF generation (50+ tables)
- Parallel QC & validation checks
- Multiple imputation runs
- Bootstrap / resampling analyses
- Independent program replication

5.2x Speedup • 4–6 Optimal Cores • 42→8s Saved

The Integrated Pipeline | All Five Technologies Working Together

phuse

connect·share·advance

1 `{arrow}`

INPUT: ADLB.sas7bdat (too large for RAM)

Streams SAS in chunks → Parquet partitioned by PARAMCD. Never loads full file. One-time conversion. 3x smaller.

INGEST

2 `{duckdb}`

INPUT: Parquet on disk (30 parameter folders)

SQL engine: window functions, CASE WHEN, LAG(), RANK(), multi-table JOINs that arrow cannot handle.

TRANSFORM

3 `{data.table}`

INPUT: Filtered result in R memory

Lightning-fast in-memory joins, TEAE flags, aggregations. 15.9x faster than dplyr. setkey() joins.

COMPUTE

4 `{furrr}`

INPUT: Single-parameter function defined

future_map() runs all 30 parameters in parallel. 8 cores × 5.2x faster. 150s → 29s.

PARALLELIZE

5 `{bench}`

INPUT: Parallel + sequential results available

mark(check=TRUE) validates parallel = sequential. Proves correctness. Regulatory audit evidence.

VALIDATE

⚡ INGEST → TRANSFORM → COMPUTE → PARALLELIZE → VALIDATE ⚡

Validation & Best Practices — Regulatory Considerations

connect·share·advance

Key Validation Requirements

Ensuring optimized R code meets regulatory standards for clinical trial analysis

Deterministic Results

All optimization techniques produce identical statistical outputs. Only speed and memory differ.

Random Seed Handling

Document seed handling for parallel processing to ensure reproducibility across runs.

Output Verification

Cross-validate optimized vs. baseline outputs using dataCompareR or equivalent tools.

Documentation Standards

Maintain audit trails; align with ICH E9 and CDISC validation guidelines.

Common Pitfalls to Avoid

- Premature optimization — profile first, optimize only where it matters
- Over-optimization — don’t sacrifice readability for marginal gains
- Ignoring validation — always verify outputs match baseline before deployment
- Parallel overhead — not beneficial for small datasets or simple operations
- Skipping documentation — always record baselines and validation evidence

Conclusion & Strategic Recommendations

phuse

connect·share·advance

Implementation Roadmap Summary

Modern R delivers 10–100x performance improvements for clinical trial programming — production-ready for regulated pharmaceutical environments with appropriate validation.

⚡ 10–100x Performance • Production-Ready • Regulatory Compliant

Recommended Implementation Path

Immediate — Transparent Optimization

For existing dplyr codebases — 11x speedup, minimal code changes.

`{dtplyr}`

Strategic — Maximum Performance

For performance-critical programs — 15–30x speedup, 75% memory savings.

`{data.table}`

Future-Proof — Beyond Memory Limits

For growing data volumes — analyze datasets 10–100x larger than RAM.

`{arrow} & {duckdb}`

Batch — Parallel Processing

For TLF generation and validation — near-linear scaling, 5x speedup.

`{furrr}`

References

[1] Dowle M, Srinivasan A (2024). data.table: Extension of data.frame. R package v1.15.0. CRAN.

[2] Richardson N, et al. (2024). arrow: Integration to 'Apache Arrow'. R package v14.0.0. CRAN.

[3] Raasveldt M, Mühleisen H (2024). duckdb: DBI Package for DuckDB. R package v0.9.2. CRAN.

[4] Barrett M, Vaughan D (2023). dtplyr: Data Table Back-End for dplyr. R package v1.3.1. CRAN.

[5] Vaughan D, Dancho M (2023). furrr: Apply Mapping Functions in Parallel using Futures. CRAN.

[6] Hester J (2023). bench: High Precision Timing of R Expressions. v1.1.3. CRAN.

[7] CDISC Analysis Data Model (ADaM) Implementation Guide, Version 1.3. CDISC, 2021.

[8] FDA (2020). Study Data Technical Conformance Guide. Technical Specifications Document.

[9] Wickham H, Bryan J (2023). R Packages (2e). https://r-pkgs.org

Questions & Discussion

Thank you for your time and attention!