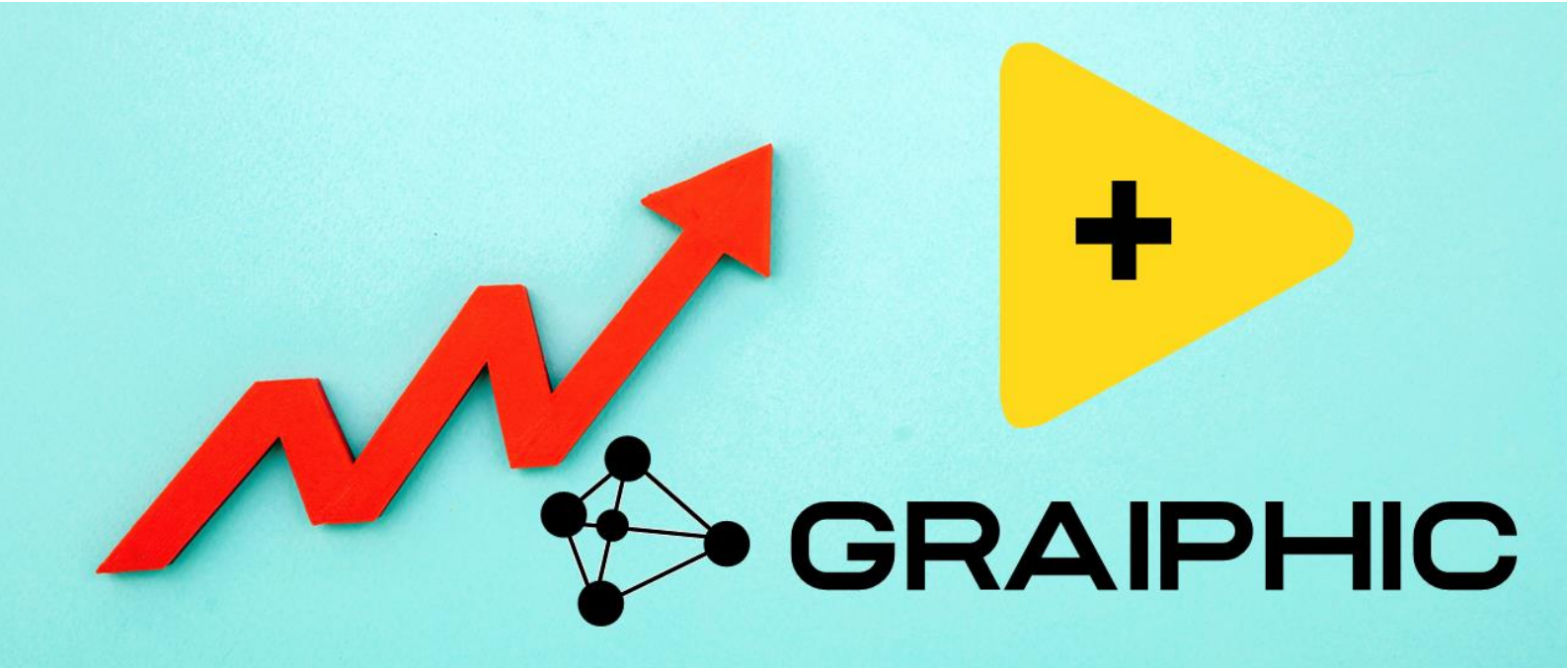
# Benchmarking the Future: Comparing LabVIEW GPU Toolkits

## CuLab, G2CPU, and the Graiphic Accelerator

ACC

G2CPU

CuLab



# Table of Contents

# Versioning

This document is subject to version control to ensure full traceability of changes. Each update is recorded with its author, date, and a short description of the modifications.

| Version | Date | Author | Organization | Change Description |
|---|---|---|---|---|
| 1.0 | 2025/10/15 | Youssef Menjour | Graiphic | First publication of the Benchmarking Whitepaper |
| 1.1 | 2025/11/07 | Youssef Menjour | Graiphic | Added DirectML execution provider |
| | | | | |
| | | | | |

# Introduction

Everyone in the LabVIEW community agrees on one thing: **LabVIEW is the best tool in the world for testing and measurement.**

So, we asked ourselves a simple question → *what if we used LabVIEW itself to test and measure the performance of LabVIEW acceleration toolkits?*

That's exactly what this benchmark does. Using pure LabVIEW, we designed a reproducible test environment to measure and compare the real-world efficiency of different GPU and CPU acceleration frameworks all within the ecosystem they were built for.

This benchmark study aims to provide a clear and reproducible comparison between several LabVIEW-based acceleration frameworks and the standard CPU execution path. Specifically, we evaluate **Graiphic Accelerator Toolkit**, **CuLab**, **C2GPU**, and **native LabVIEW CPU execution** across a series of representative computational workloads.

The objective is not limited to raw performance measurement. Instead, this study seeks to analyze how each framework behaves under various computational patterns from standard arithmetic operations to matrix multiplications and complex-number processing in order to better understand their strengths, limitations, and integration trade-offs within the LabVIEW environment.

Four test cases are considered:

1. **GEMM (General Matrix Multiplication):** a canonical linear algebra benchmark designed to measure computational throughput and memory efficiency.
2. **Standard Arithmetic Execution:** a baseline test representing typical scalar and vector arithmetic operations in LabVIEW without GPU acceleration.
3. **Complex Number Computation:** a specialized test assessing how each framework manages complex arithmetic, a domain where certain ONNX operators still lack native support. For this case, **Graiphic** introduces custom complex nodes that encode real and imaginary parts along an additional tensor dimension, a functional though not yet fully optimized approach.
4. **Signal Processing Benchmark:** this application-oriented test reads data from a file (which could also originate from a live sensor), applies a signal processing pipeline that includes an FFT, logarithmic, additive, and divisive operations, and finally visualizes the output waveform. This benchmark highlights real-world use cases where **Graiphic Accelerator Toolkit** can accelerate end-to-end dataflow execution directly within the LabVIEW environment.

All experiments are performed on **NVIDIA hardware**, and the **Graiphic Accelerator Toolkit** is tested under both **CUDA** and **TensorRT Execution Providers** to illustrate how runtime compilation and low-level kernel optimizations impact performance and determinism.

The results presented in the following sections focus on **execution time**, **numerical precision**, and **resource utilization**, providing insights into the performance-to-integration ratio of each approach and guiding developers in selecting the most suitable acceleration strategy for their LabVIEW-based AI or data processing applications.

# Test Environment

All benchmarks were executed under the following hardware and software configuration:

- **OS:** Windows 11
- **CPU:** Intel® Core™ i9-10850K CPU @ 3.60 GHz
- **GPU:** NVIDIA GeForce RTX 3060
- **LabVIEW Version:** LabVIEW 2025 Q3
- **Date of Execution:** October 15, 2025

This configuration was chosen as a balanced, widely available workstation setup representative of real-world development and deployment environments. It provides sufficient computational headroom to evaluate both CPU and GPU performance while maintaining reproducibility across different toolkits.

## What We Test

The benchmark compares multiple LabVIEW-based acceleration toolkits and execution providers using the latest stable releases available at the time of testing. All frameworks were evaluated under identical conditions on the same hardware and LabVIEW version.

## Tested Configurations:

| Toolkit Name | Version | Repository / Source | Company | Execution Mode(s) Tested |
|---|---|---|---|---|
| Graiphic Accelerator Toolkit | 1.0.0.6 | SOTA (Graiphic) | Graiphic | GPU (CUDA)<br>GPU (TensorRT)<br>GPU (DirectML) |
| CuLab – GPU Accelerated Toolkit | 4.1.2.80 | VIPM (JKI) | Ngene | GPU (CUDA) |
| G2CPU – GPU and CPU HPC Toolkit | 1.6.0.15 | VIPM (JKI) | Natan Biesmans | GPU (CUDA) |

## Runtime and Driver Stack:

- **CUDA:** 12.9
- **TensorRT:** 10.13.3.9
- **DirectML:** 1.15.4.0

These configurations were selected to represent both *graph-compiled* (TensorRT) and *kernel-executed* (CUDA) acceleration modes, along with other popular LabVIEW GPU frameworks, in order to provide a comprehensive performance and compatibility comparison.

## Data and Resources Availability

All benchmark results, LabVIEW VIs, and related materials used in this study are publicly available on **Graiphic's official GitHub repository**:

👉 https://github.com/Graiphic/whitepapers

The repository includes the complete test setup, datasets, and example projects to allow anyone to reproduce the results or extend the benchmarks with additional toolkits and configurations.

# Benchmark 1 – The GEMM Processing

## Description

This benchmark represents a computational workflow consisting of a **matrix multiplication (GEMM)**, followed by a few arithmetic operations, and ending with a scalar output. The objective is to measure raw computational throughput and the efficiency of memory transfers between LabVIEW and the underlying GPU or CPU backends.

## Results

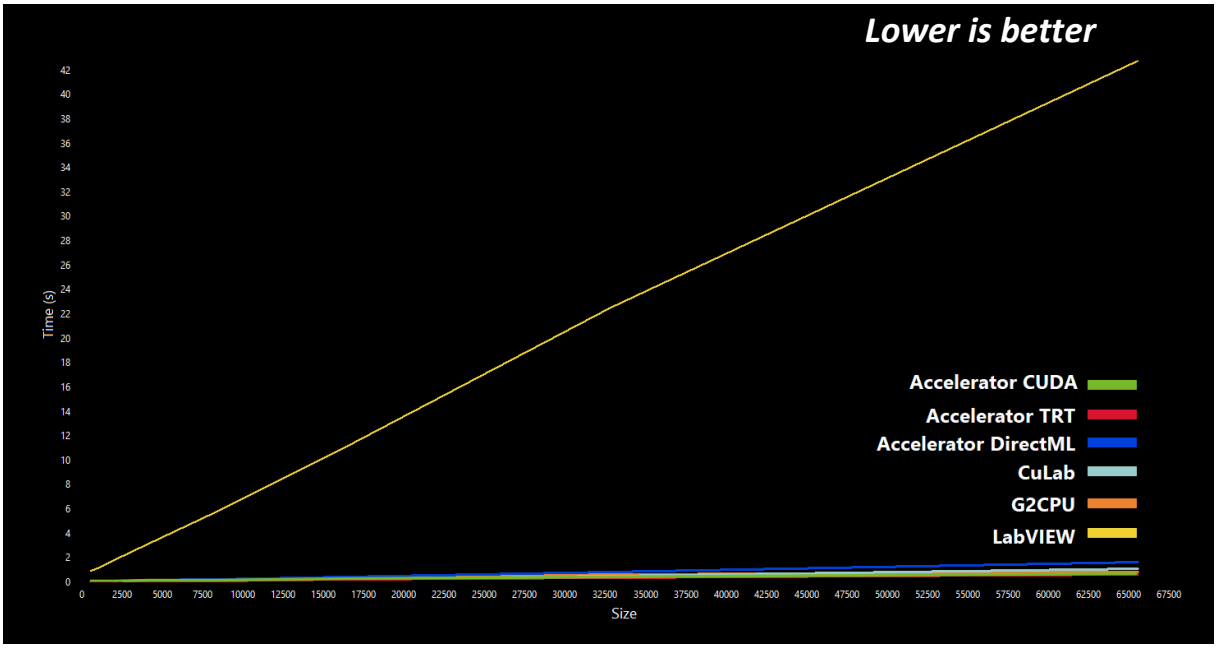



***Figure 1*** *– Full-scale comparison including LabVIEW CPU execution (LabVIEW clearly stands out due to CPU-only processing)*

The results show a **significant performance gap** between standard LabVIEW CPU execution and GPU-accelerated toolkits.

The CPU execution curve demonstrates a **linear progression of processing time with array size**, confirming that computation scales predictably but lacks the parallel

acceleration provided by GPU architectures. This clearly illustrates the **necessity of GPU utilization** for high-performance numerical processing.

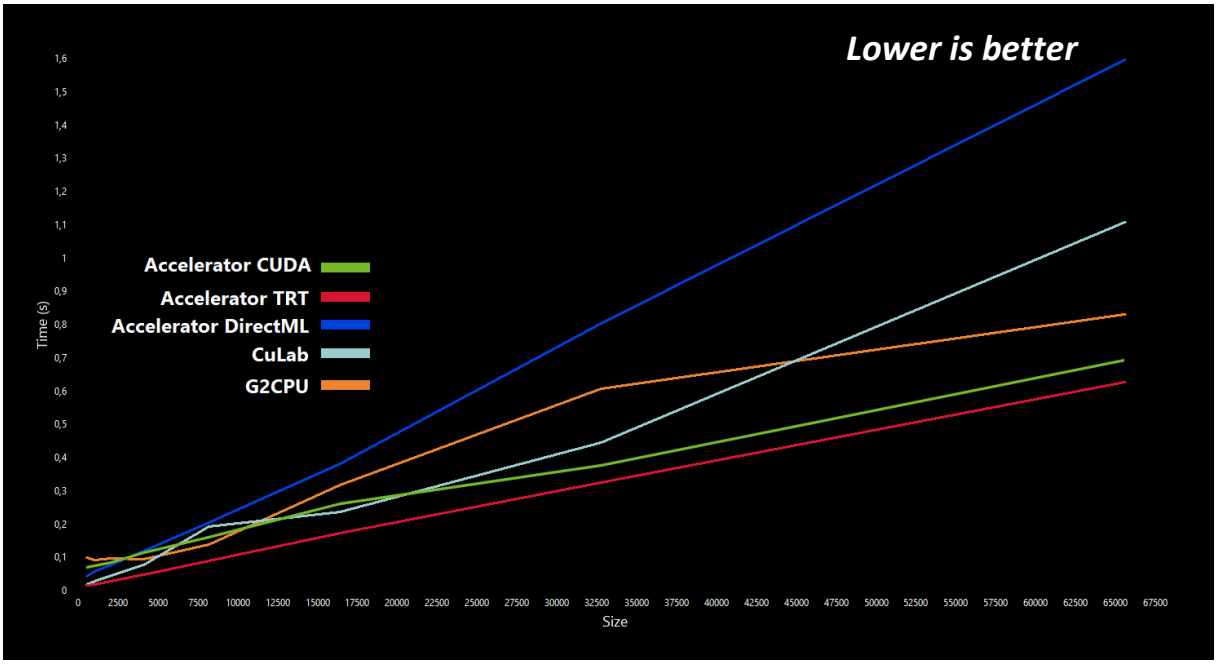



*Figure 2 – Zoomed-in comparison of GPU-based toolkits*
*(Highlights performance differences between Accelerator Toolkit, CuLab, and G2CPU)*

## Analysis

When comparing GPU toolkits only, the **TensorRT Execution Provider Provider (Graiphic Accelerator Toolkit)** achieves the **best performance**, followed closely by the **CUDA Execution Provider (Graiphic Accelerator Toolkit)**.

In contrast, **CuLab** and **G2CPU** exhibit noticeably higher processing times, with performance ratios ranging between **1.3× and 2× slower** compared to Graiphic's implementation.

This performance advantage is primarily explained by the **compilation strategy** adopted by Graiphic.

While CuLab and G2CPU rely on **DLL-based calls** that require multiple LabVIEW↔library transitions during execution, the Graiphic Accelerator Toolkit **compiles the computational graph ahead of time**, reducing overhead and enabling optimized GPU kernel scheduling.

This precompiled graph execution allows TensorRT and CUDA to fully exploit low-level hardware optimizations, leading to faster and more deterministic results.

# Benchmark 2 – Arithmetic Operations

## Description

This benchmark measures the performance of basic arithmetic operations applied iteratively on increasing array sizes.

Each iteration executes a loop containing six consecutive operations — **addition, negation, multiplication, and division** — followed by a summation of all elements to produce a scalar output.

This test is designed to evaluate how efficiently each toolkit handles **element-wise operations** and memory management under lightweight but repetitive workloads.

## Results

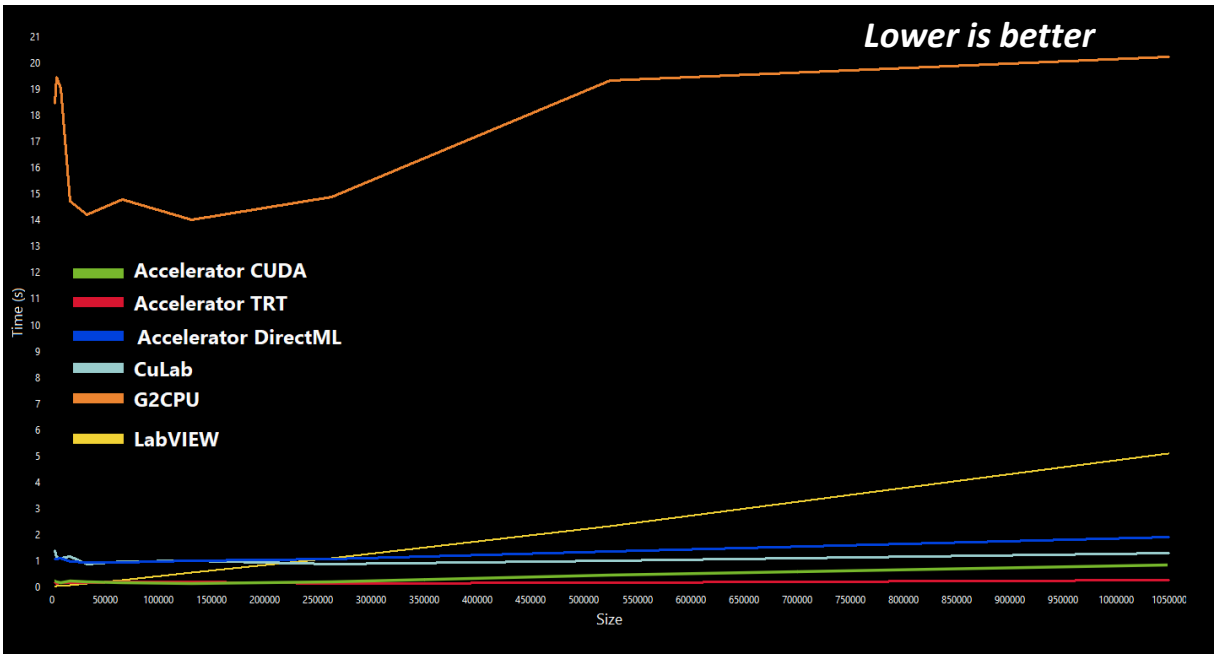



*Figure 3 – Performance comparison for repeated arithmetic operations (Add/Neg/Mul/Div) (Execution time vs. input array size)*

## Analysis

The results show an even more pronounced gap between the **Graiphic Accelerator Toolkit** and other toolkits than in the GEMM benchmark.

The **TensorRT Execution Provider (Graiphic Accelerator)** completely outperforms all alternatives, achieving up to **5× faster performance than CuLab** and up to **40× faster than G2CPU**, despite the simplicity of the operations involved.

The **CUDA Execution Provider** from Graiphic also maintains a strong advantage, performing between **2× and 5× faster than CuLab** and **8× to 40× faster than G2CPU**, depending on the input size.

These impressive gains highlight the efficiency of **Graiphic's compilation-based execution model**. Unlike DLL-based toolkits (CuLab and G2CPU), which rely on repeated LabVIEW–DLL transitions during runtime, the Graiphic Accelerator Toolkit **precompiles the computation graph** and executes it natively on the GPU, minimizing memory transfers and kernel launch overhead.

In this context of simple, high-frequency arithmetic loops, the benefits of this approach are amplified:

- **TensorRT** leverages aggressive operator fusion and kernel scheduling, resulting in minimal latency and maximal throughput.
- **CUDA EP** benefits from efficient memory reuse and optimized compute dispatch.
- Conversely, **CuLab** and **G2CPU** incur high overheads from repeated calls through DLL layers, explaining their comparatively slower and less scalable behavior.

Overall, the benchmark demonstrates that even for **elementary arithmetic workloads**, the choice of execution model (compiled vs interpreted) plays a decisive role — with Graiphic's TensorRT pipeline delivering **the highest efficiency and consistency** across all tested array sizes.

# Benchmark 3 – Complex Number Computation

## Description

This benchmark introduces a deliberate challenge: the use of **complex numbers** within LabVIEW GPU-accelerated computation.

The purpose is to evaluate how each toolkit behaves when faced with operations that are **not natively supported by the ONNX specification**, which serves as the abstraction layer for the Graiphic Accelerator Toolkit.

Complex arithmetic is highly relevant in industrial and scientific applications — for example, in signal processing, impedance analysis, or electromagnetics — yet the current ONNX standard does not provide full native support for complex datatypes.

To overcome this limitation, **Graiphic implemented its own workaround**, representing complex numbers by **adding an additional tensor dimension** and adjusting the computational graph to handle real and imaginary parts explicitly. While this approach ensures functional correctness, it is **not inherently optimized**, since it adds extra memory and indexing overhead.

## Results

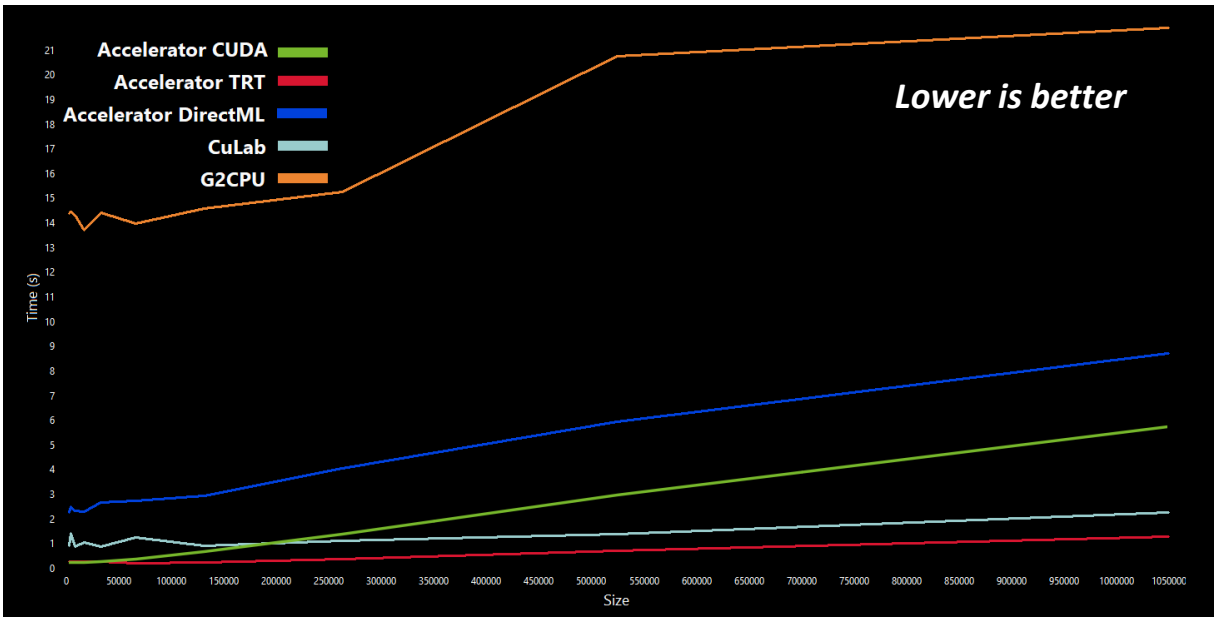



*Figure 4 – Complex Number Benchmark: execution time vs. input size*

## Analysis

This benchmark is particularly interesting because it **deliberately puts Graiphic at a disadvantage**.
Despite this, the results demonstrate remarkable robustness of the Graiphic toolkits:

- The **Accelerator CUDA Execution Provider** shows a **predictable linear progression** in processing time as data size increases, consistent with the expected behavior due to the added tensor dimension.
- The **TensorRT Execution Provider**, however, **maintains superior performance**, remaining faster even under these non-optimized conditions. This confirms TensorRT's ability to efficiently fuse and optimize graph operations, even when the model structure is artificially expanded.

In contrast, the other toolkits — **CuLab** and **G2CPU** — exhibit slower and less stable scaling, particularly at higher data sizes.

This test highlights two key takeaways:

1. **Not everything is perfect yet.** The handling of complex datatypes in ONNX remains a technical gap that impacts all graph-based runtimes.
2. **Graiphic's architectural control** allows for deep customization and rapid evolution. Because the toolkit is fully designed and maintained in-house, Graiphic can adapt its compiler and runtime to overcome limitations rather than depend on opaque third-party libraries.

Looking ahead, Graiphic plans to **extend ONNX with native complex number abstraction** and to **contribute this feature to ONNX Runtime**, reinforcing its active participation in the open-source AI ecosystem.

# Benchmark 4 – Signal Processing Application

## Description

This benchmark replicates a real-world use case commonly encountered in test and measurement applications.

It reads **signal data from a file** (which could equivalently come from a live sensor acquisition), then applies a **signal processing pipeline** including an **FFT**, **logarithmic scaling**, **addition**, and **division** operations.

Finally, the processed data are displayed as a waveform on a LabVIEW graph, mimicking a typical monitoring or measurement system.

## Results

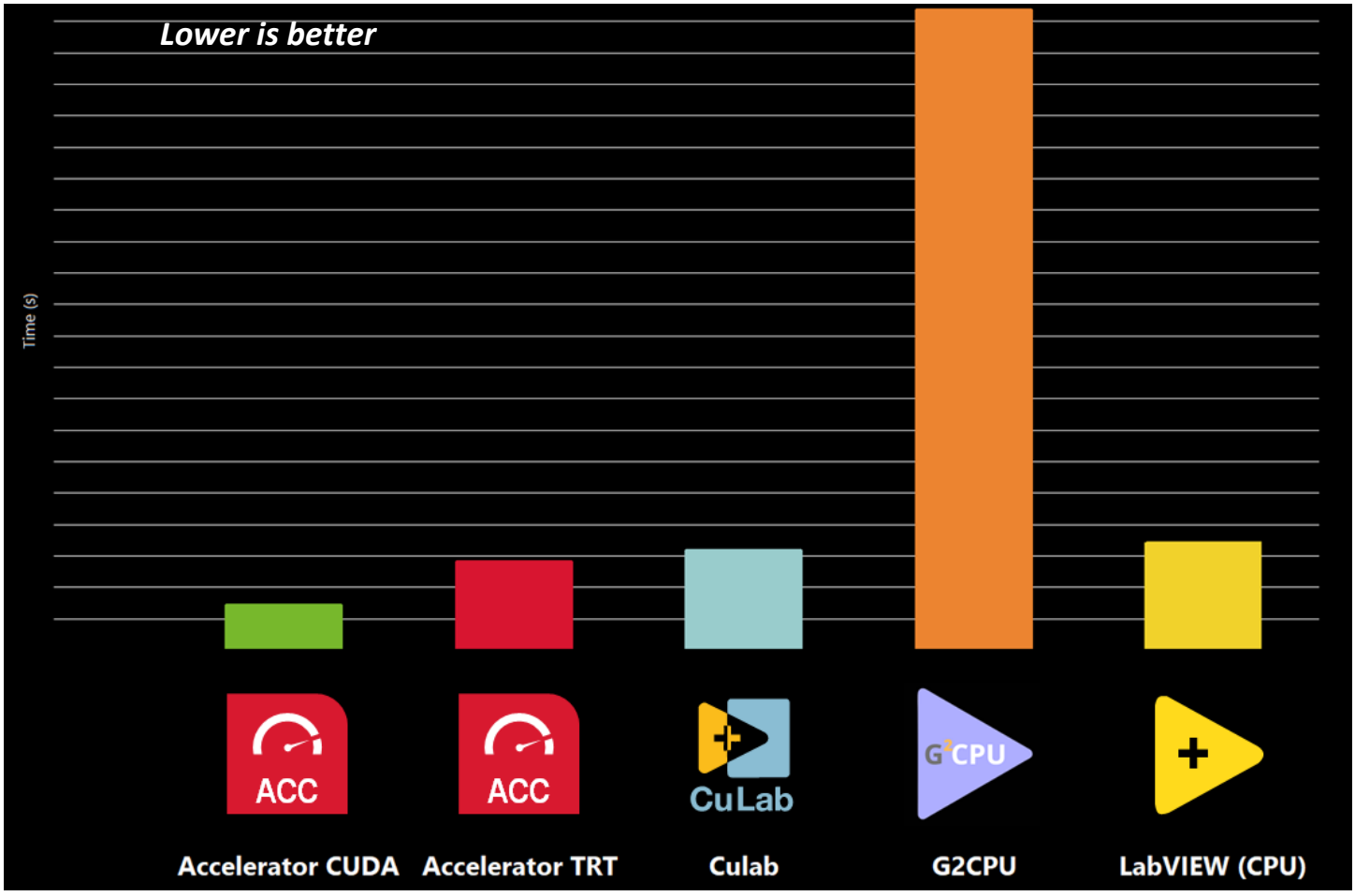



*Figure 5 – Processing time comparison across toolkits for signal processing workload (FFT + arithmetic operations on small sensor-like data chunks)*

## Analysis

Unlike the previous synthetic benchmarks (GEMM, arithmetic loops, complex tensors), this test focuses on **application-level behavior** using relatively **small data blocks** — here around **32k samples** per read operation.

In such conditions, the **performance gap between CPU and GPU** naturally shrinks. The workload is not large enough to fully exploit GPU parallelism, and the overhead of GPU kernel setup and data transfer becomes relatively significant compared to the computation time itself.

Consequently, **LabVIEW CPU execution** remains **competitive** in this specific scenario, showing performance close to GPU-based toolkits. This aligns with expectations: for **low data throughput or lightly parallel tasks**, the CPU can outperform or match GPU execution due to its lower initialization latency.

Still, the **Graiphic Accelerator Toolkit** (both CUDA and TensorRT versions) **outperforms all other GPU toolkits**, confirming the efficiency of its compiled execution model and optimized memory management even at small scales.

**CuLab** and **G2CPU** perform adequately but exhibit higher variance and slightly slower execution times due to their runtime overhead and DLL-based data exchange.

# Another Comparative Overview

| Feature | CuLab | G2CPU | Graiphic Accelerator Toolkit |
|---|---|---|---|
| Execution Model | Immediate (per node) | Immediate (ArrayFire abstraction) | Compiled graph (ONNX Runtime) |
| Multi-provider Support | CUDA only | CPU, CUDA, OpenCL | CPU, CUDA, TensorRT, DirectML, others |
| Control Flow Support (If, Loop, Scan) | No | No | Yes |
| File Format | None (VI-level) | None (ArrayFire abstraction) | ONNX |
| Graph Reusability | Limited | Limited | Full (portable artifact) |
| Fallback Mechanism | None | Manual | Automatic via Execution Providers |
| Integration Depth | Library binding | Abstract API | Native compiler bridge |
| Performance Tuning | Per node | Per node | End-to-end optimization |
| Philosophy | Accelerate LabVIEW functions | Abstract backend selection | Unify AI, logic, and execution |

This table illustrates the essential difference between past and present. CuLab and G2CPU extend LabVIEW through external computation. The Graiphic Accelerator Toolkit extends LabVIEW itself turning its diagrams into compilable graphs that can execute anywhere ONNX Runtime or MLIR compilers exist.

# Call for Funding: Why Industry Should Invest in GO HW

Graiphic has built the first end-to-end ecosystem where AI, logic, and hardware orchestration live inside a single ONNX graph. We call it Graph Orchestration Hardware.

We are now opening a Call for Funding to accelerate the roadmap of Graiphic GO HW Project.

**Why Invest?**

- Strategic Advantage: Gain early access to the first universal cockpit that unifies AI + hardware orchestration across CPUs, GPUs, FPGAs, NPUs, and SoCs.
- Portability & Standards: Ensure your hardware, SDKs, and platforms are natively supported in a framework that is becoming the de facto open standard.
- Energy & Efficiency: Join the revolution of Green AI by design. GO HW introduces forensic-grade energy metrics and optimization directly inside ONNX graphs.
- Market Reach: From Raspberry Pi to NVIDIA Jetson, from industrial PLCs to cloud servers, GO HW runs everywhere, and your technology can be part of it.

**What We Offer**

- Co-development opportunities with our engineering team.
- Early integration of your platforms and SDKs into GO HW.
- Joint visibility in international standardization efforts (ONNX, DARPA, Horizon Europe, ADRA).
- Shared benchmarking and open-source dissemination to establish your technology as a leader in AI orchestration.

**How to Engage**

Graiphic is actively seeking:

- Equity investors ready to support our growth.
- Industrial sponsors willing to co-fund R&D and test benches.
- Strategic partners (hardware vendors, system integrators, large OEMs) who want their platforms at the heart of the future ONNX Hardware ecosystem.

Join us in shaping the universal cockpit for AI.

Contact: funding@graiphic.io | www.graiphic.io