DotCompute

GPU Acceleration Framework for .NET

High-Performance SIMD, CUDA, OpenCL & Metal Compute

.NET 9 • Native AOT • Source Generators • Roslyn Analyzers

Executive Overview

Version 0.5.3

A production-grade .NET GPU acceleration framework delivering CPU 3.7x (SIMD) and GPU 21–92x (CUDA) speedups with native AOT support, source generators, and multi-backend compute.

Built with C# 13 & .NET 9 • 4-layer architecture • 5 compute backends

Open-source (MIT License) • NuGet packages available

Repository: github.com/mivertowski/DotCompute

Documentation: mivertowski.github.io/DotCompute

Contact: michael.ivertowski@ch.ey.com

Contents

1 Executive Summary

DotCompute is a production-grade GPU acceleration framework for .NET 9+, designed to bring high-performance parallel computing to the .NET ecosystem. It provides a unified programming model across CPU (AVX2/AVX512/NEON SIMD), CUDA, OpenCL, and Metal backends — enabling developers to write compute kernels once and execute them on any supported hardware with automatic backend selection and optimization.

Why DotCompute?

- **Massive speedups** — CPU SIMD delivers 3.7x, CUDA delivers 21–92x over baseline on NVIDIA RTX hardware (Compute Capability 8.9).
- **Native AOT compatible** — sub-10 ms startup with no runtime code generation; source generators handle all compile-time codegen.
- **Multi-backend compute** — write once, run on CPU, CUDA, OpenCL, or Metal with automatic backend selection.
- **Developer tooling** — 12 Roslyn diagnostic rules, 5 automated code fixes, and `[Kernel]` attribute-driven source generation provide real-time IDE feedback.
- **Production infrastructure** — memory pooling (90% allocation reduction), P2P transfers, dependency injection integration, and plugin system with hot-reload capability.

1.1 At a Glance

92x peak GPU speedup (CUDA)	**5** compute backends	**<10 ms** Native AOT startup

2 Compute Backends

DotCompute provides five compute backends, each optimized for specific hardware targets while sharing a unified kernel programming model.

Backend	Status	Speedup	Capabilities
CPU	Production	3.7x	AVX2, AVX512, and ARM NEON SIMD intrinsics; thread-based persistent kernel execution.
CUDA	Production	21–92x	Compute Capability 5.0–8.9; CUBIN (CC $\geq$ 7.0) and PTX compilation; P2P transfers; NCCL multi-GPU; Ring Kernels.
OpenCL	Experimental	—	Cross-platform support for NVIDIA, AMD, Intel, ARM Mali, and Qualcomm Adreno GPUs.
Metal	Feature-Complete	—	Apple GPU via Metal Performance Shaders; MSL kernel translation; Ring Kernel execution; memory pooling.
ROCm	Placeholder	—	AMD GPU backend planned for future development.

An **Adaptive Backend Selector** uses ML-powered heuristics to automatically choose the optimal backend based on data size, kernel complexity, hardware availability, and historical profiling data.

3 Kernel Programming Model

DotCompute provides a modern, attribute-driven kernel programming model that eliminates boilerplate and enables compile-time optimization through source generators.

3.1 Kernel Authoring

```
[Kernel]
public static void VectorAdd(
ReadOnlySpan<float> a,
ReadOnlySpan<float> b,
Span<float> result)
{
int idx = Kernel.ThreadId.X;
if (idx < result.Length)
result[idx] = a[idx] + b[idx];
}
```

The `[Kernel]` attribute triggers source generation at compile time, producing optimized wrappers for each target backend. No runtime reflection or code generation is required, preserving full Native AOT compatibility. The pipeline is: `C# Source` → `Roslyn Analysis` (12 rules) → `Source Generation` → `Backend Wrappers`.

3.2 Kernel System Components

Component	Description
IKernelCompiler	Backend-specific kernel compilation (PTX, SPIR-V, MSL, SIMD).
KernelDefinition	Metadata describing kernel signature, thread dimensions, and compilation options.
ICompiledKernel	Ready-to-execute kernel with launch configuration and resource bindings.
CompilationOptions	Optimization level, debug info, target compute capability, and backend-specific flags.

3.3 Roslyn Analyzers (DC001–DC012)

Rule	Severity	Description
DC001–DC004	Error	Kernel signature validation (return type, parameters, static)
DC005–DC008	Warning	Memory access patterns and bounds checking
DC009–DC010	Warning	Thread divergence and synchronization issues
DC011–DC012	Info	Performance hints and optimization suggestions

All rules include **5 automated code fixes** for one-click resolution of common kernel authoring issues directly in the IDE.

4 Ring Kernel System

The Ring Kernel system implements a persistent GPU computation model based on the actor pattern, enabling long-lived GPU kernels that process messages with sub-microsecond serialization. The pipeline flows from host application through a message bridge and MemoryPack serialization ($<$100 ns) to a persistent or event-driven GPU kernel.

4.1 Implementation Phases

Phase	Status	Capabilities
Phase 1: MemoryPack	Complete	Auto-discovery of `[MemoryPackable]` types via Roslyn; batch CUDA codegen for 10+ message types; $<$100 ns serialization.
Phase 2: CPU Ring	Complete	Thread-based persistent kernels; message queue bridge; echo mode with intelligent transformation; full lifecycle management.
Phase 3: CUDA Ring	94.3%	End-to-end GPU message processing; 6-stage compilation pipeline; $\sim$1.24 μs serialization.
Phase 4: Multi-GPU	Complete	P2P memory transfer infrastructure; cross-GPU barrier foundations; coordination primitives for multi-device setups.
Phase 5: Observability	Complete	OpenTelemetry integration; Prometheus-compatible metrics; health checks; fault tolerance with Polly resilience policies.

4.2 GPU Compute Primitives

API	Details
GPU Timing	1 ns resolution on CC 6.0+, 4 calibration strategies, per-kernel profiling.
Barrier API	ThreadBlock, Grid, Warp, and Named barriers with <20 ns overhead.
Memory Ordering	3 consistency models (relaxed, acquire-release, sequential) with proper fence semantics.
Unified Buffer	Cross-device memory abstraction with automatic transfer management and pooled allocation.

5 Memory Management

DotCompute provides a layered memory subsystem that minimizes allocation overhead and automates cross-device transfers:

- **UnifiedBuffer<T>**: Generic, type-safe buffer abstraction that works identically across CPU, CUDA, OpenCL, and Metal with automatic lifecycle management.
- **Memory pooling**: Size-class bucketed allocation pool reduces GC pressure and achieves 90% fewer allocations.
- **Automatic transfers**: The runtime transparently moves data between host (pinned) and device (VRAM) as kernels require.
- **P2P transfers**: Direct GPU-to-GPU copies for multi-device workloads without host memory round-trips.
- **Bounds validation**: Debug-mode bounds checking in kernels catches out-of-range accesses before they corrupt memory.

6 Developer Tooling & Runtime

DotCompute integrates deeply into the .NET developer workflow through source generators, Roslyn analyzers, and runtime infrastructure.

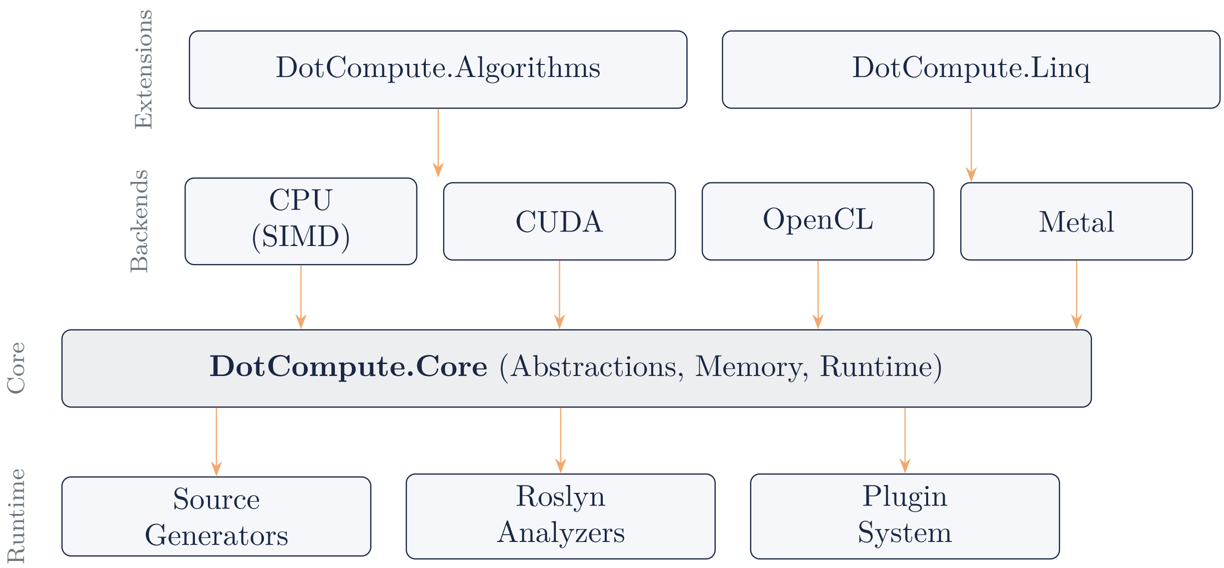
Component	Description
Source Generators	Roslyn-based compile-time codegen from `[Kernel]` attributes; produces backend-specific wrappers with zero runtime reflection.
DI Integration	Full Microsoft.Extensions.DependencyInjection support with `AddDotCompute()` service registration.
Plugin System	Hot-reload capable plugin architecture for custom backends and kernel providers.
Orchestrator	`IComputeOrchestrator` manages kernel compilation, execution, and resource lifecycle.
Debug Service	`IKernelDebugService` enables CPU vs. GPU result validation for cross-backend debugging.
Adaptive Optimizer	ML-powered `AdaptiveBackendSelector` learns optimal backend assignments from runtime profiling.

7 LINQ Extensions

DotCompute extends .NET LINQ with GPU-accelerated query execution, enabling familiar `Select`, `Where`, and `Aggregate` operations to transparently compile to GPU kernels. Adjacent operations are automatically fused into single kernel launches to reduce overhead.

Operation	Status	Details
Select / Map	Complete	Element-wise transformations with GPU codegen.
Where / Filter	Complete	Predicate evaluation with stream compaction.
Aggregate / Reduce	Complete	Parallel reduction with configurable operators.
Kernel Fusion	Complete	Adjacent operations merged into single kernel launch.
Join	Planned	Hash-based GPU join for relational operations.
GroupBy	Planned	GPU-accelerated grouping with aggregation.
OrderBy	Planned	Parallel sort (bitonic/radix) on GPU.

8 Architecture



9 Use Cases

Primary Use Cases

- **Scientific Computing** — GPU-accelerated linear algebra, matrix operations, and numerical simulations with 21–92x speedups on NVIDIA hardware.
- **Machine Learning Inference** — High-throughput tensor operations for .NET ML pipelines with automatic SIMD/CUDA backend selection.
- **Data Processing Pipelines** — GPU-accelerated LINQ extensions for large-scale data transformations with kernel fusion optimization.
- **Real-Time Signal Processing** — Ring Kernel actor model for persistent GPU computation with sub-microsecond message serialization.
- **Financial Computing** — Monte Carlo simulations, risk calculations, and portfolio optimization leveraging GPU parallelism.
- **Cross-Platform GPU Compute** — Write-once kernels targeting Windows (CUDA/OpenCL), macOS (Metal), and Linux (CUDA/OpenCL) from a single .NET codebase.

10 Getting Started

10.1 Installation

```
# Add DotCompute NuGet packages
dotnet add package DotCompute.Core
dotnet add package DotCompute.Backend.Cpu
dotnet add package DotCompute.Backend.Cuda   # NVIDIA GPU
dotnet add package DotCompute.Backend.Metal  # Apple GPU
```

10.2 Quick Start

```
# Build the solution
dotnet build DotCompute.sln -configuration Release

# Run all tests (recommended -- auto-configures WSL2)
./scripts/run-tests.sh DotCompute.sln -configuration Release

# Run specific test categories
./scripts/run-tests.sh DotCompute.sln -filter "Category=Unit"
./scripts/run-tests.sh DotCompute.sln -filter "Category=Hardware"
```

10.3 Requirements

- .NET 9.0 SDK or later (C# 13 language features)
- Visual Studio 2022 17.8+ or VS Code with C# Dev Kit
- CUDA Toolkit 12.0+ (for GPU support)
- NVIDIA GPU with Compute Capability 5.0+ (for CUDA tests)

Links & Resources

Repository	https://github.com/mivertowski/DotCompute
Documentation	https://mivertowski.github.io/DotCompute/
NuGet	DotCompute.Core, DotCompute.Backend.Cpu, DotCompute.Backend.Cuda

Contact: michael.ivertowski@ch.ey.com