Armen Melkumyan • 1st
Technical / Solutions Architect
5mo •

.NET Performance with DirectCompute and ComputeSharp⚡

DirectCompute gives .NET developers the power to leverage GPU acceleration for general-purpose computations using the DirectX API. With the help of ComputeSharp, a modern library designed to execute DirectCompute shaders in C#, you can offload parallelizable tasks to the GPU, achieving remarkable performance improvements for heavy computations.

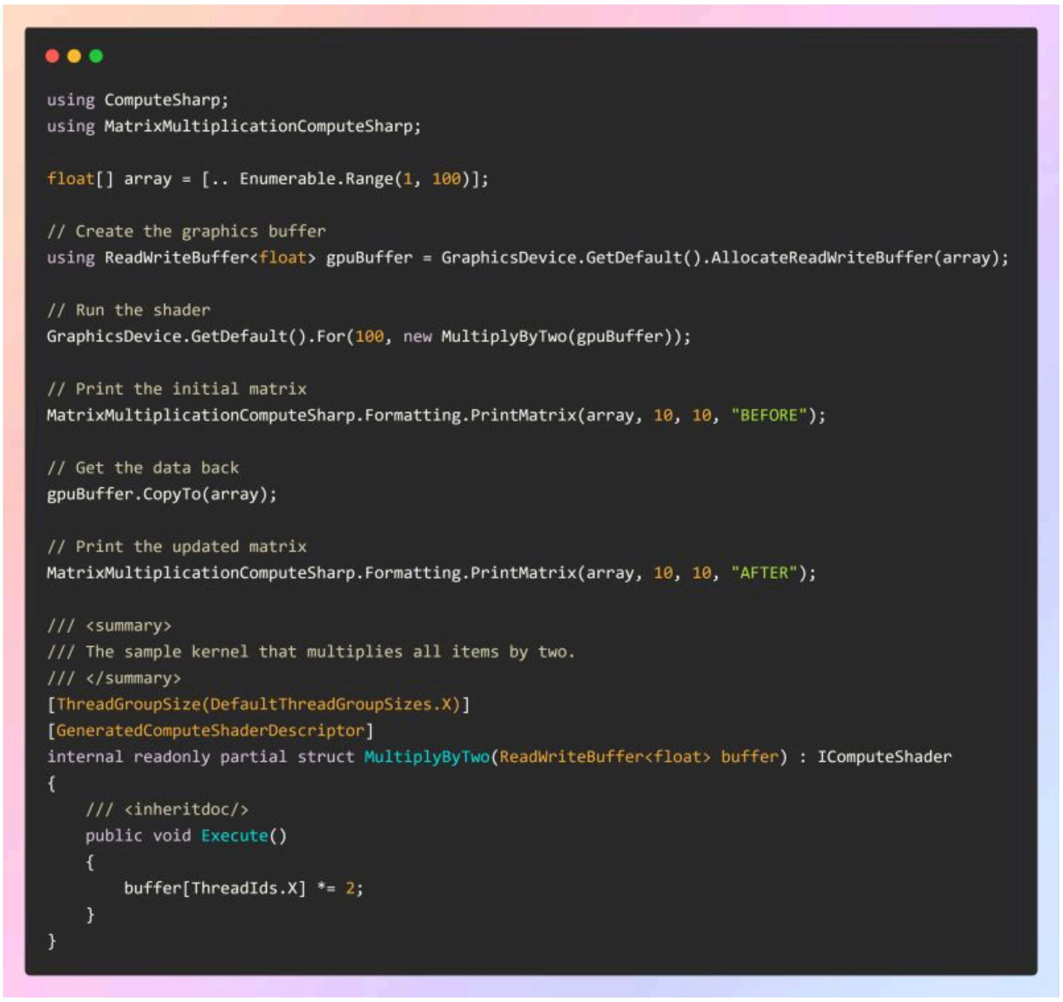
In this example, we'll walk through a simple but effective use case: multiplying every element in an array by two. Here’s how this GPU-accelerated process works:

Steps:
1. Allocate GPU memory: We use the `GraphicsDevice` to allocate GPU buffers to hold the data that will be processed.
2. Run the shader: The shader processes each element in parallel, multiplying every array element by two using the GPU.
3. Retrieve the data: Once processed, the data is transferred back to the CPU for further use or display.
4. Print the results: We print the matrix before and after GPU processing, demonstrating the effect of the shader.

Why GPU Acceleration Matters:
For compute-heavy and parallelizable tasks (e.g., matrix operations, image processing, or scientific simulations), the GPU’s architecture can provide significant speed improvements compared to traditional CPU-based methods. Leveraging ComputeSharp makes this more accessible than ever in the .NET ecosystem.

💡 Tip: GPU acceleration isn't just for graphics! It's a powerful tool for any computation-heavy workload where parallel processing can make a difference. Start small with simple array operations, then explore more complex use cases like machine learning, large-scale data processing, or scientific computing.

#GPUComputing #DirectCompute #ComputeSharp #HighPerformanceComputing #ParallelProcessing #CodeSnippets #GPUEngineering



```csharp
using ComputeSharp;
using MatrixMultiplicationComputeSharp;

float[] array = [.. Enumerable.Range(1, 100)];

// Create the graphics buffer
using ReadWriteBuffer<float> gpuBuffer = GraphicsDevice.GetDefault().AllocateReadWriteBuffer(array);

// Run the shader
GraphicsDevice.GetDefault().For(100, new MultiplyByTwo(gpuBuffer));

// Print the initial matrix
MatrixMultiplicationComputeSharp.Formatting.PrintMatrix(array, 10, 10, "BEFORE");

// Get the data back
gpuBuffer.CopyTo(array);

// Print the updated matrix
MatrixMultiplicationComputeSharp.Formatting.PrintMatrix(array, 10, 10, "AFTER");

/// <summary>
/// The sample kernel that multiplies all items by two.
/// </summary>
[ThreadGroupSize(DefaultThreadGroupSizes.X)]
[GeneratedComputeShaderDescriptor]
internal readonly partial struct MultiplyByTwo(ReadWriteBuffer<float> buffer) : IComputeShader
{
    /// <inheritdoc/>
    public void Execute()
    {
        buffer[ThreadIds.X] *= 2;
    }
}
```

57 2 comments 1 repost