.NET Conf China 2025

改变世界 改变自己

2025 年 11 月 30 日 | 中国 上海

.NET Conf China 2025
改变世界 改变自己

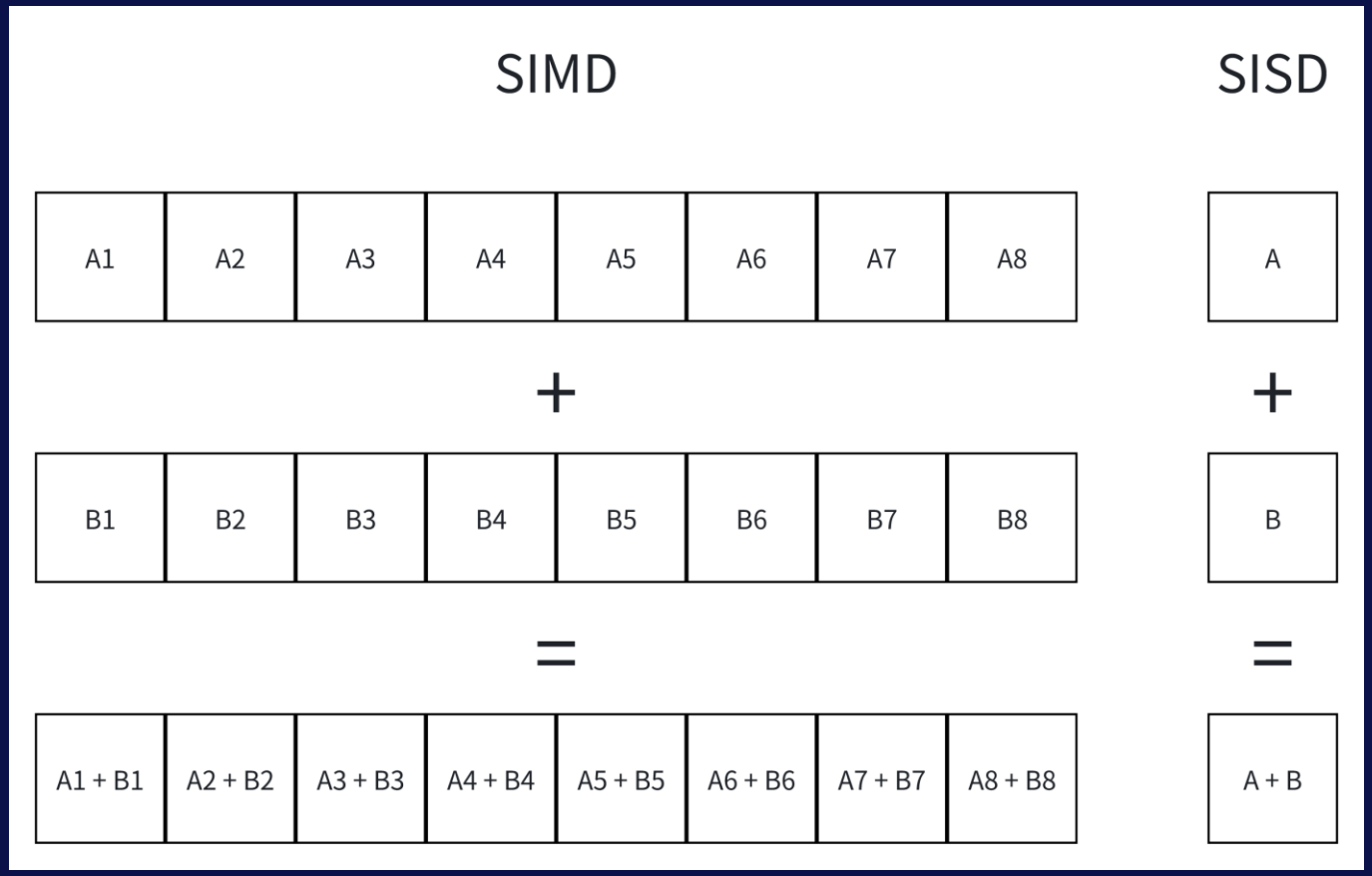
如何在 .NET 中使用 SIMD

黄凯华
iHerb .NET 后端开发

SIMD（Single Instruction, Multiple Data） 译为 单指令多数据，是一种并行计算技术，允许单条指令同时对多个数据元素进行操作，从而提高计算效率。

与 SIMD 相对的是 SISD（Single Instruction, Single Data，单指令单数据），即每条指令只处理一个数据元素。

如果我们要对两组数组进行加法运算，传统方法（SISD）是逐个元素相加，而使用 SIMD 技术，可以一次性将多个元素加载到向量寄存器中，并执行单一的加法指令，从而显著提高计算效率。



Program.cs

```cs
using System.Runtime.Intrinsics;
using BenchmarkDotNet.Attributes;
using BenchmarkDotNet.Running;

[MemoryDiagnoser]
1 usage
public class SimdBenchmark
{
    private float[] _arrA;
    private float[] _arrB;
    private float[] _resultArray;
    private readonly int _dataSize = 1_000_000;

    [GlobalSetup]
    public void Setup()
    {
        var random = new Random();
        _arrA = new float[_dataSize];
        _arrB = new float[_dataSize];
        _resultArray = new float[_dataSize];

        for (int i = 0; i < _dataSize; i++)
        {
            _arrA[i] = (float)random.NextDouble() * 10f;
            _arrB[i] = (float)random.NextDouble() * 10f;
        }
    }
```

```
28      [Benchmark]
29      public void NormalAdd()
30      {
31          for (int i = 0; i < _dataSize; i++)
32          {
33              _resultArray[i] = _arrA[i] + _arrB[i];
34          }
35      }
```

```
37      [Benchmark]
38      public void SimdAdd()
39      {
40          // 每次处理 4 个元素
41          int simdLength = Vector128<float>.Count; // 4
42          int i = 0;
43
44          // 处理可被 SIMD 整除的部分
45          for (; i <= _dataSize - simdLength; i += simdLength)
46          {
47              // 表示从数组的第 i 个位置开始加载数据到向量中，每次加载 4 个 float
48              var va :Vector128<float> = Vector128.LoadUnsafe(ref _arrA[i]);
49              var vb :Vector128<float> = Vector128.LoadUnsafe(ref _arrB[i]);
50              (va + vb).CopyTo(_resultArray, i);
51          }
52
53          // 处理尾部不足 4 个的元素
54          for (; i < _dataSize; i++)
55          {
56              _resultArray[i] = _arrA[i] + _arrB[i];
57          }
58      }
```

```
BenchmarkDotNet v0.15.6, macOS Sequoia 15.7.2 (24G325) [Darwin 24.6.0]
Apple M2 Max, 1 CPU, 12 logical and 12 physical cores
.NET SDK 10.0.100
  [Host]     : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT armv8.0-a
  DefaultJob : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT armv8.0-a


| Method    | Mean     | Error   | StdDev  | Allocated |
|---------- |---------:|--------:|--------:|----------:|
| NormalAdd | 894.9 us | 3.99 us | 3.11 us |         - |
| SimdAdd   | 302.0 us | 3.10 us | 2.59 us |         - |
```

```
BenchmarkDotNet v0.15.6, Windows 11 (10.0.26100.7019/24H2/2024Update/HudsonValley)
AMD Ryzen 5 9600X 3.90GHz, 1 CPU, 12 logical and 6 physical cores
.NET SDK 10.0.100
  [Host]     : .NET 10.0.0 (10.0.0, 10.0.25.52411), X64 RyuJIT x86-64-v4
  DefaultJob : .NET 10.0.0 (10.0.0, 10.0.25.52411), X64 RyuJIT x86-64-v4
```

Method	Mean	Error	StdDev	Allocated
NormalAdd	594.0 us	14.69 us	43.10 us	-
SimdAdd	174.0 us	2.84 us	2.66 us	-

System.Runtime.Intrinsics 命名空间

.NET 为我们提供了下面三个命名空间来使用 SIMD 技术：

- System.Runtime.Intrinsics ：包含用于创建和传递各种大小和格式的寄存器状态的类型。
- System.Runtime.Intrinsics.X86 ：包含特定于 x86/x64 架构的 SIMD 指令集的类型。
- System.Runtime.Intrinsics.Arm ：包含特定于 ARM 架构的 SIMD 指令集的类型。

System.Runtime.Intrinsics 命名空间

System.Runtime.Intrinsics 命名空间中定义了表示不同大小向量的结构体和提供创建及操作这些向量的静态类。

结构体	描述
Vector64<T>	表示指定数值类型的 64 位向量，该向量适用于并行算法的低级别优化。
Vector128<T>	表示指定数值类型的 128 位向量，该向量适用于并行算法的低级别优化。
Vector256<T>	表示指定数值类型的 256 位向量，该向量适用于并行算法的低级别优化。
Vector512<T>	表示指定数值类型的 512 位向量，该向量适用于并行算法的低级别优化。

System.Runtime.Intrinsics 命名空间

静态类	描述
Vector64	提供静态方法的集合，用于在 64 位向量上创建、操作和以其他方式操作。
Vector128	提供静态方法集合，用于在 128 位向量上创建、操作和以其他方式操作。
Vector256	提供静态方法集合，用于在 256 位向量上创建、操作和以其他方式操作。
Vector512	提供静态方法的集合，用于在 512 位向量上创建、操作和以其他方式操作。

System.Runtime.Intrinsics 命名空间

System.Runtime.Intrinsics.X86 和 System.Runtime.Intrinsics.Arm 命名空间中定义了特定于各自架构的 SIMD 指令集的类，这些类提供了访问底层硬件 SIMD 指令的能力。

常见的指令集类例如：

类型	描述
Sse	提供对 x86/x64 SSE 指令集的访问。
Sse2	提供对 x86/x64 SSE2 指令集的访问。
Avx	提供对 x86/x64 AVX 指令集的访问。
Avx2	提供对 x86/x64 AVX2 指令集的访问。
AdvSimd	提供对 ARM Advanced SIMD 指令集的访问。

更详细的列表可以参考官方文档：
https://learn.microsoft.com/zh-cn/dotnet/api/system.runtime.intrinsics.x86

https://learn.microsoft.com/zh-cn/dotnet/api/system.runtime.intrinsics.arm

如何理解向量的大小

向量的大小（如 64 位、128 位、256 位、512 位）指的是向量寄存器能够容纳的数据总位数。每个向量寄存器可以存储多个数据元素，这些数据元素的类型和数量取决于向量的大小和数据类型的位数。

例如 Vector128<float>，它表示一个 128 位的向量寄存器，可以存储 4 个 32 位的浮点数（因为 128 / 32 = 4）。

如果是用来存储 64 位的双精度浮点数（double），则 Vector128<double> 可以存储 2 个双精度浮点数（因为 128 / 64 = 2）。

如何理解向量的大小

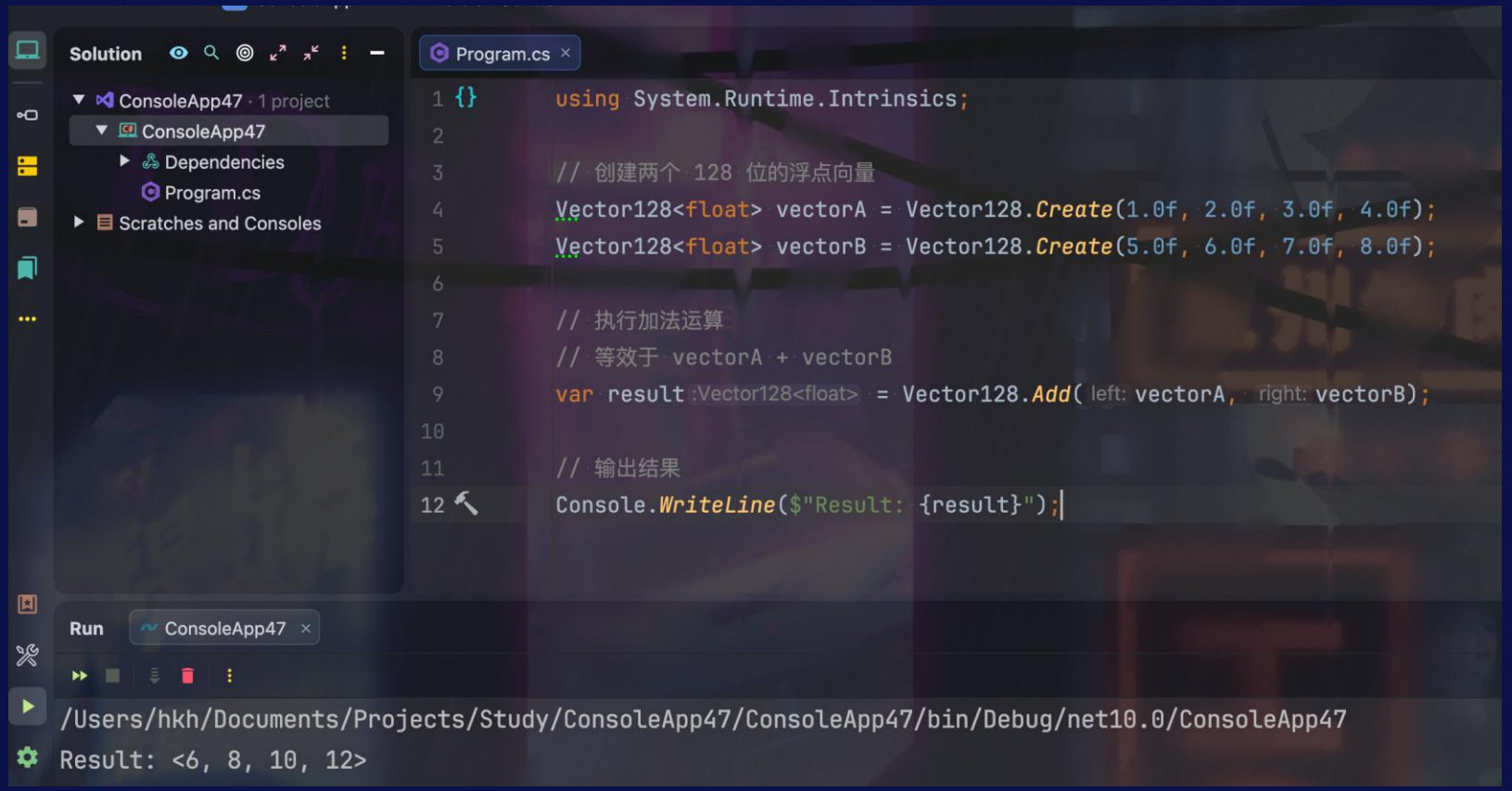

```
using System.Runtime.Intrinsics;

// 创建一个 128 位的向量，存储 16 个 8 位的 字节
Vector128<byte> vectorByte = Vector128.Create((byte)1, (byte)2, (byte)3, (byte)4,
(byte)5, (byte)6, (byte)7, (byte)8,
(byte)9, (byte)10, (byte)11, (byte)12,
(byte)13, (byte)14, (byte)15, (byte)16);

// 创建一个 128 位的向量，存储 4 个 32 位的 浮点数
Vector128<float> vectorFloat = Vector128.Create(1.0f, 2.0f, 3.0f, 4.0f);

// 创建一个 256 位的向量，存储 8 个 32 位的 浮点数
Vector256<float> vector256Float = Vector256.Create(1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f, 7.0f, 8.0f);

// 创建一个 128 位的向量，存储 2 个 64 位的 双精度浮点数
Vector128<double> vectorDouble = Vector128.Create(1.0, 2.0);

// 创建一个 256 位的向量，存储 4 个 64 位的 双精度浮点数
Vector256<double> vector256Double = Vector256.Create(1.0, 2.0, 3.0, 4.0);
```

跨平台实现方式

.NET 的 SIMD 提供了跨平台的实现方式。无论是在 x86/x64 还是 ARM 架构上，.NET 都会根据运行时环境自动选择合适的 SIMD 指令集来执行向量化操作。

VectorXXX 为我们提供了一组静态方法，用于创建和操作向量。例如，Vector128.Add 方法用于对两个 128 位向量执行加法运算。

我们也可以直接使用运算符号来进行向量运算，例如 +、-、*、/ 等。VectorXXX<T> 结构体重载了这些运算符，使得向量运算更加直观和简洁。



跨平台实现方式

可以使用：

VectorXXX.IsHardwareAccelerated 检查某个宽度是可以硬件加速；

VectorXXX<T>.IsSupported 检查特定宽度+类型组合是否可用。

跨平台实现方式


```
Console.WriteLine(Vector512.IsHardwareAccelerated ? "Vector512 支持硬件加速" :
"Vector512 不支持硬件加速");
Console.WriteLine(Vector256.IsHardwareAccelerated ? "Vector256 支持硬件加速" :
"Vector256 不支持硬件加速");
Console.WriteLine(Vector128.IsHardwareAccelerated ? "Vector128 支持硬件加速" :
"Vector128 不支持硬件加速");

Console.WriteLine(Vector128<int>.IsSupported ? "Vector128<int> 支持" :
"Vector128<int> 不支持");
Console.WriteLine(Vector256<int>.IsSupported ? "Vector256<int> 支持" :
"Vector256<int> 不支持");
Console.WriteLine(Vector512<int>.IsSupported ? "Vector512<int> 支持" :
"Vector512<int> 不支持");

Vector512<int> vectorA = Vector512.Create(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16);
Vector512<int> vectorB = Vector512.Create(16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6,
5, 4, 3, 2, 1);

Vector512<int> result = Vector512.Add(vectorA, vectorB);

Console.WriteLine("result: " + result);
```

跨平台实现方式


```
Vector512 不支持硬件加速
Vector256 不支持硬件加速
Vector128 支持硬件加速
Vector128<int> 支持
Vector256<int> 支持
Vector512<int> 支持
result: <17, 17, 17, 17, 17, 17, 17, 17, 17, 17,
17, 17, 17, 17, 17, 17>
```

跨平台实现方式


```
Console.WriteLine(Vector128<bool>.IsSupported ? "Vector128<bool> 支持" : "Vector128<bool> 不支持");

bool[] arr = [true, false, true, false, true, false, true, false];

Vector128<bool> v = Vector128.LoadUnsafe(ref arr[0]);
```

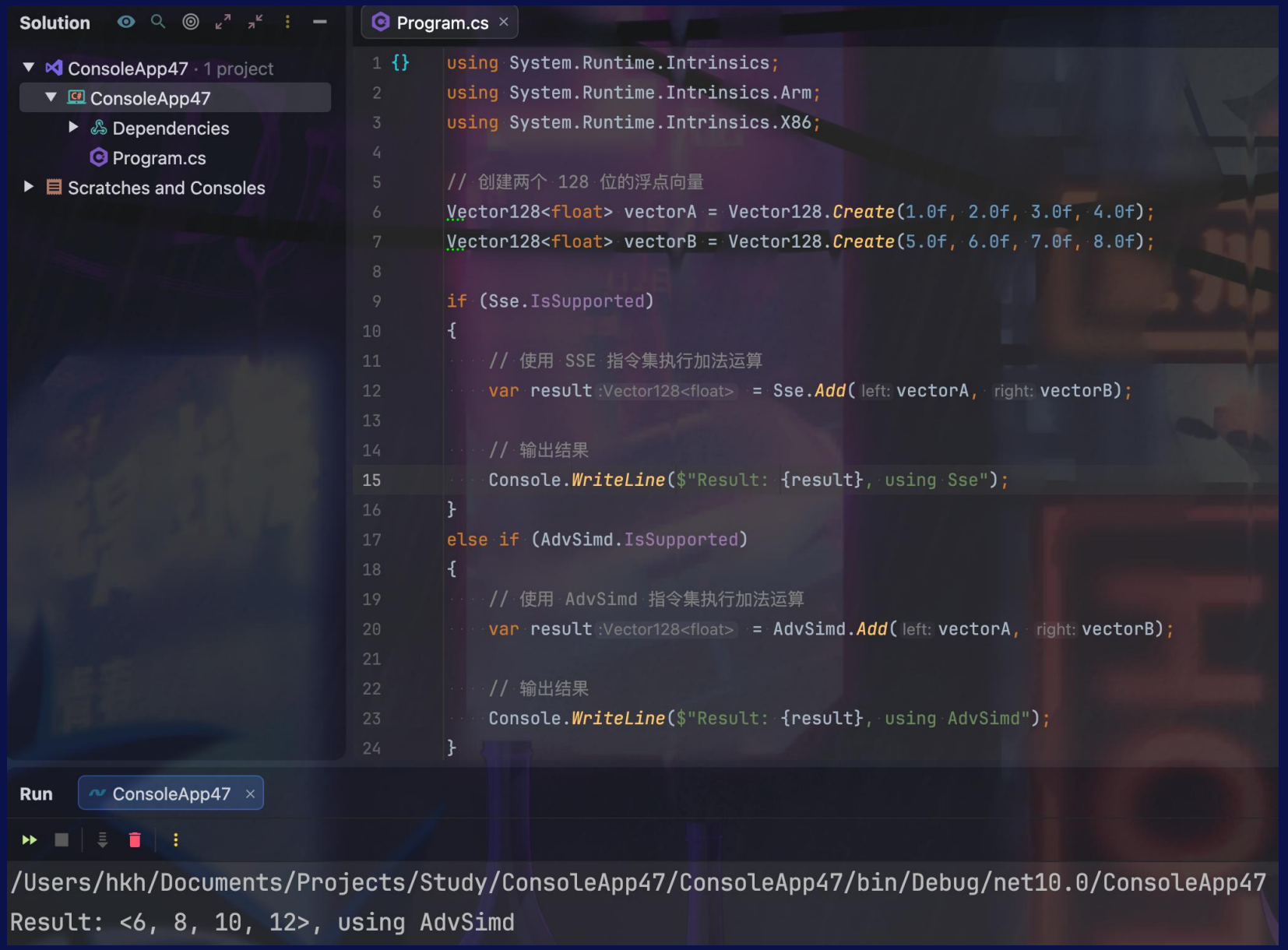
```
Vector128<bool> 不支持
Unhandled exception. System.NotSupportedException: Specified type is not
supported
```

SIMD 指令集的使用

在使用 SIMD 指令集之前，通常需要检查当前平台是否支持特定的指令集。可以通过调用指令集类的 IsSupported 属性来进行检查。例如：

```
using System.Runtime.Intrinsics.X86;

Console.WriteLine(Sse.IsSupported ? "SSE 指令集受支持" : "SSE 指令集不受支持");
```

SIMD 指令集的使用


Solution
ConsoleApp47 · 1 project
ConsoleApp47
Dependencies
Program.cs
Scratches and Consoles
Program.cs
using System.Runtime.Intrinsics;
using System.Runtime.Intrinsics.Arm;
using System.Runtime.Intrinsics.X86;
// 创建两个 128 位的浮点向量
Vector128<float> vectorA = Vector128.Create(1.0f, 2.0f, 3.0f, 4.0f);
Vector128<float> vectorB = Vector128.Create(5.0f, 6.0f, 7.0f, 8.0f);
if (Sse.IsSupported)
{
// 使用 SSE 指令集执行加法运算
var result :Vector128<float> = Sse.Add(left: vectorA, right: vectorB);
// 输出结果
Console.WriteLine($"Result: {result}, using Sse");
}
else if (AdvSimd.IsSupported)
{
// 使用 AdvSimd 指令集执行加法运算
var result :Vector128<float> = AdvSimd.Add(left: vectorA, right: vectorB);
// 输出结果
Console.WriteLine($"Result: {result}, using AdvSimd");
}
Run
ConsoleApp47
/Users/hkh/Documents/Projects/Study/ConsoleApp47/ConsoleApp47/bin/Debug/net10.0/ConsoleApp47
Result: <6, 8, 10, 12>, using AdvSimd

选择合适的向量创建方式

在创建向量时，可以根据具体需求选择不同的创建方式：

- VectorXXX.Create：适用于创建包含特定值的向量，适合少量元素的初始化。
- VectorXXX.Load 和 VectorXXX.LoadUnsafe：适用于从数组中加载数据到向量，适合大量数据的处理。Load 方法的参数是 指针 类型，需要使用 unsafe 代码块，而 LoadUnsafe 方法则可以直接传递 托管指针（ref，常说的引用传递）。

选择合适的向量创建方式


```
[Benchmark]
public void Vector128CreateAdd()
{
    int simdLength = Vector128<float>.Count; // 4
    int i = 0;

    for (; i <= _dataSize - simdLength; i += simdLength)
    {
        var va :Vector128<float> = Vector128.Create(_arrA[i], _arrA[i + 1], _arrA[i + 2], _arrA[i + 3]);
        var vb :Vector128<float> = Vector128.Create(_arrB[i], _arrB[i + 1], _arrB[i + 2], _arrB[i + 3]);
        (va + vb).CopyTo(_resultArray, i);
    }

    for (; i < _dataSize; i++)
    {
        _resultArray[i] = _arrA[i] + _arrB[i];
    }
}
```

选择合适的向量创建方式


```
[Benchmark]
public unsafe void Vector128LoadAdd()
{
    int simdLength = Vector128<float>.Count;
    int i = 0;

    fixed (float* pA = _arrA)
    fixed (float* pB = _arrB)
    {
        for (; i <= _dataSize - simdLength; i += simdLength)
        {
            var va :Vector128<float> = Vector128.Load( source: pA + i);
            var vb :Vector128<float> = Vector128.Load( source: pB + i);
            (va + vb).CopyTo(_resultArray, i);
        }
    }

    for (; i < _dataSize; i++)
    {
        _resultArray[i] = _arrA[i] + _arrB[i];
    }
}
```

选择合适的向量创建方式


```
[Benchmark]
public void Vector128LoadUnsafeAdd()
{
    int simdLength = Vector128<float>.Count; // 4
    int i = 0;

    for (; i <= _dataSize - simdLength; i += simdLength)
    {
        var va :Vector128<float> = Vector128.LoadUnsafe(ref _arrA[i]);
        var vb :Vector128<float> = Vector128.LoadUnsafe(ref _arrB[i]);
        (va + vb).CopyTo(_resultArray, i);
    }

    for (; i < _dataSize; i++)
    {
        _resultArray[i] = _arrA[i] + _arrB[i];
    }
}
```

选择合适的向量创建方式


```
BenchmarkDotNet v0.15.6, macOS Sequoia 15.7.2 (24G325) [Darwin 24.6.0]
Apple M2 Max, 1 CPU, 12 logical and 12 physical cores
.NET SDK 10.0.100
  [Host]     : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT armv8.0-a
  DefaultJob : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT armv8.0-a


| Method                | Mean     | Error   | StdDev  | Allocated |
|---------------------- |---------:|--------:|--------:|----------:|
| Vector128CreateAdd    | 489.3 us | 4.99 us | 4.17 us |         - |
| Vector128LoadAdd      | 186.3 us | 2.56 us | 2.27 us |         - |
| Vector128LoadUnsafeAdd | 302.7 us | 2.83 us | 2.36 us |         - |
```

System.Numerics 命名空间

基于 System.Runtime.Intrinsics，.NET 还提供了别的更高级别的 SIMD 支持。

比如在 System.Numerics 这个命名空间提供了一些易于使用的类型，如 Vector<T> 和 Matrix4x4，简化了 SIMD 编程。

https://learn.microsoft.com/zh-cn/dotnet/standard/simd

Vector<T>

Vector<T> 是一个通用的向量类型，支持多种数值类型（如 int、float、double 等）。
JIT 时会根据硬件能力自动选择最佳的向量大小（如 128 位或 256 位），
从而实现跨平台的 SIMD 优化。

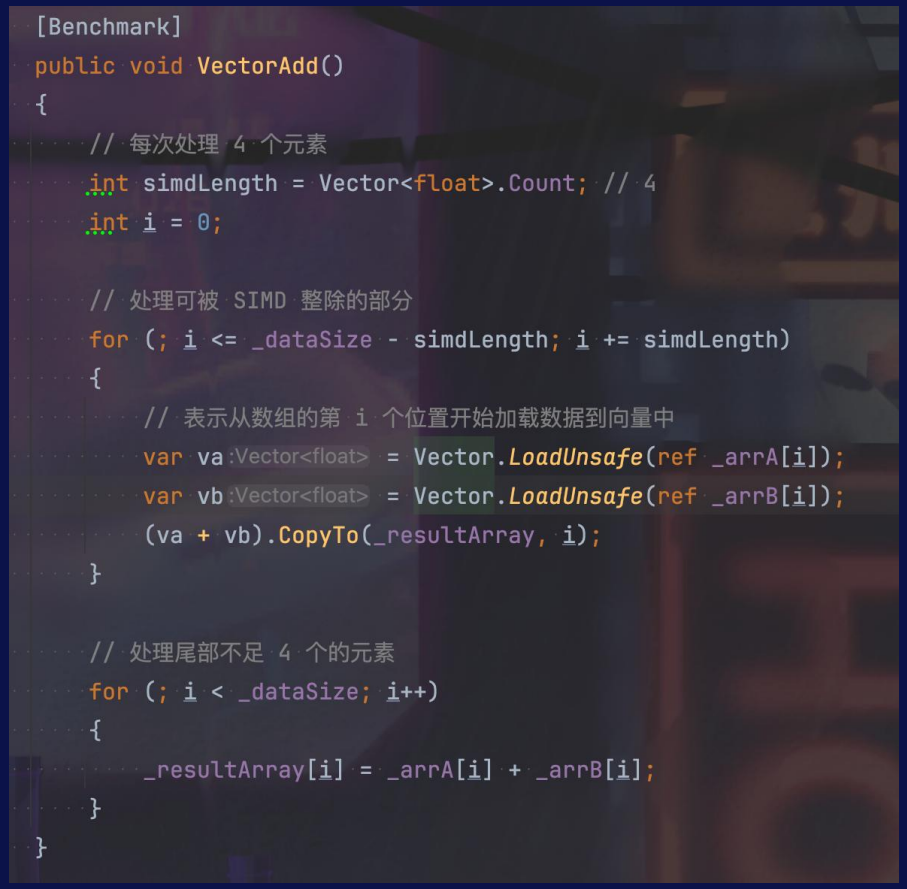
如果硬件不支持 SIMD，Vector<T> 会退化为普通的标量操作。

Vector_1.cs

src > libraries > System.Private.CoreLib > src > System > Numerics > Vector_1.cs > Vector

```
31      public readonly unsafe struct Vector<T> : ISimdVector<Vector<T>, T>, IFormattable
219
220         /// <summary>Adds two vectors to compute their sum.</summary>
221         /// <param name="left">The vector to add with <paramref name="right" />.</param>
222         /// <param name="right">The vector to add with <paramref name="left" />.</param>
223         /// <returns>The sum of <paramref name="left" /> and <paramref name="right" />.</returns>
224         [Intrinsic]
225         [MethodImpl(MethodImplOptions.AggressiveInlining)]
226         public static Vector<T> operator +(Vector<T> left, Vector<T> right)
227         {
228             Unsafe.SkipInit(out Vector<T> result);
229
230             for (int index = 0; index < Count; index++)
231             {
232                 T value = Scalar<T>.Add(left.GetElementUnsafe(index), right.GetElementUnsafe(index));
233                 result.SetElementUnsafe(index, value);
234             }
235
236             return result;
237         }
```

Vector<T>



```
[Benchmark]
public void VectorAdd()
{
    // 每次处理 4 个元素
    int simdLength = Vector<float>.Count; // 4
    int i = 0;

    // 处理可被 SIMD 整除的部分
    for (; i <= _dataSize - simdLength; i += simdLength)
    {
        // 表示从数组的第 i 个位置开始加载数据到向量中
        var va:Vector<float> = Vector.LoadUnsafe(ref _arrA[i]);
        var vb:Vector<float> = Vector.LoadUnsafe(ref _arrB[i]);
        (va + vb).CopyTo(_resultArray, i);
    }

    // 处理尾部不足 4 个的元素
    for (; i < _dataSize; i++)
    {
        _resultArray[i] = _arrA[i] + _arrB[i];
    }
}
```

```
BenchmarkDotNet v0.15.6, macOS Sequoia 15.7.2 (24G325) [Darwin
24.6.0]
Apple M2 Max, 1 CPU, 12 logical and 12 physical cores
.NET SDK 10.0.100
  [Host]     : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT
armv8.0-a
  DefaultJob : .NET 10.0.0 (10.0.0, 10.0.25.52411), Arm64 RyuJIT
armv8.0-a


| Method       | Mean     | Error    | StdDev   | Allocated |
|------------- |---------:|---------:|---------:|----------:|
| NormalAdd    | 911.2 us | 9.09 us  | 8.50 us  |         - |
| Vector128Add | 301.3 us | 1.90 us  | 1.59 us  |         - |
| VectorAdd    | 300.9 us | 2.61 us  | 2.44 us  |         - |
```

Vector2、Vector3 和 Vector4 结构体

System.Numerics 命名空间还提供了 Vector2、Vector3 和 Vector4 结构体，分别表示二维、三维和四维向量，常用于图形和物理计算中。

```
// 创建 Vector3 实例
Vector3 vector1 = new Vector3(1.0f, 2.0f, 3.0f);
Vector3 vector2 = new Vector3(4.0f, 5.0f, 6.0f);

// 向量加法
Vector3 resultAdd = Vector3.Add(vector1, vector2);
Console.WriteLine($"Addition: {resultAdd}"); // 输出: <5, 7, 9>

// 向量点乘
float dotProduct = Vector3.Dot(vector1, vector2);
Console.WriteLine($"Dot Product: {dotProduct}"); // 输出: 32

// 向量归一化
Vector3 normalized = Vector3.Normalize(vector1);
Console.WriteLine($"Normalized: {normalized}"); // 输出: <0.2672612, 0.5345225, 0.8017837>
```

Matrix2x2、Matrix3x2 和 Matrix4x4

System.Numerics 还提供了 Matrix2x2、Matrix3x2 和 Matrix4x4 结构体，用于表示二维和三维空间中的矩阵，常用于变换和投影计算。

```
Matrix4x4 matrix = Matrix4x4.CreateRotationX((float)(Math.PI / 4));
Vector3 point = new Vector3(1.0f, 0.0f, 0.0f);

// 使用矩阵变换点
Vector3 transformedPoint = Vector3.Transform(point, matrix);
Console.WriteLine($"Transformed Point: {transformedPoint}"); // 输出: <1, 0, 0>
```

其他 SIMD 的使用场景举例

Ascii.ToUpper
Ascii.ToLower

BinaryPrimitives.ReverseEndianness

Guid

.NET Conf China 2025

改变世界 改变自己

THANK YOU