ALTENAR

SIMD в .NET

Почему «быстрый» код не всегда ускоряет ваши вычисления

Кирилл Панин
Старший .NET разработчик, Altenar

Кирилл Панин

Старший .NET разработчик

5+ лет
в Backend разработке

Опыт проектирования и оптимизаций математических моделей

Altenar. Кто это?

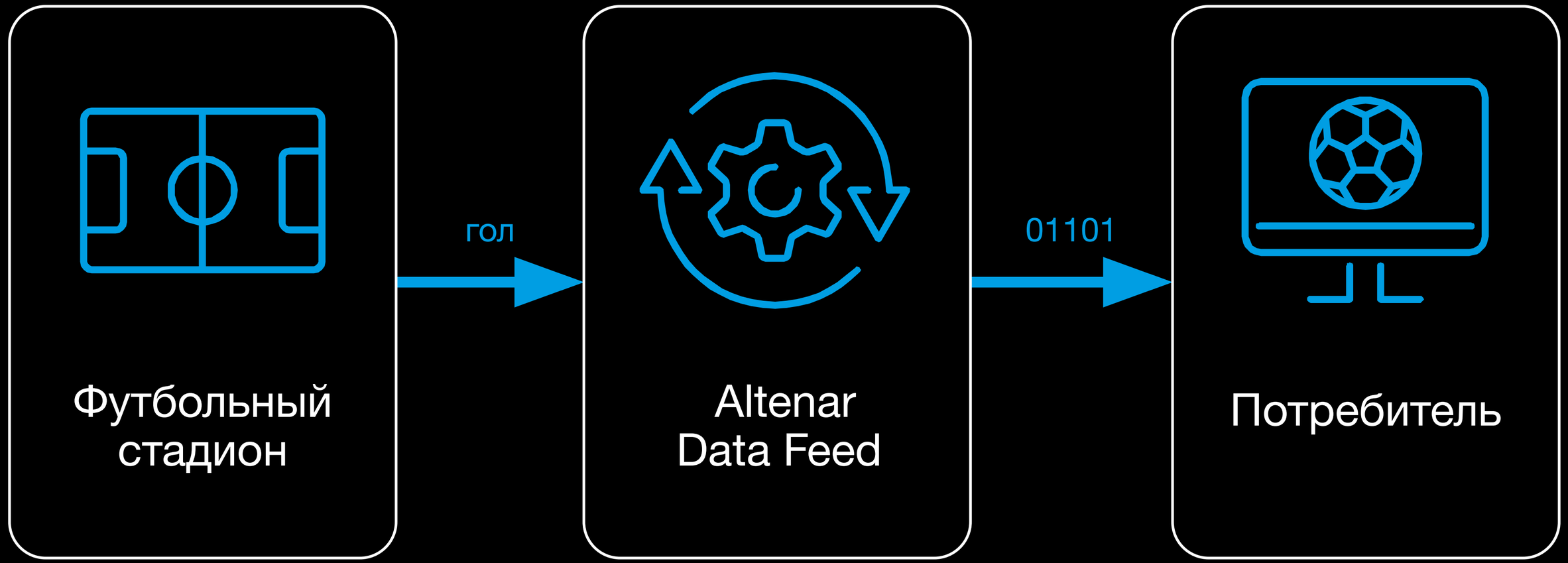
ALTENAR | LEADING SPORTSBOOK PLATFORM

10+ лет работы на международном рынке

700+ сотрудников, много удаленщиков

50+ стран, где есть клиенты компании

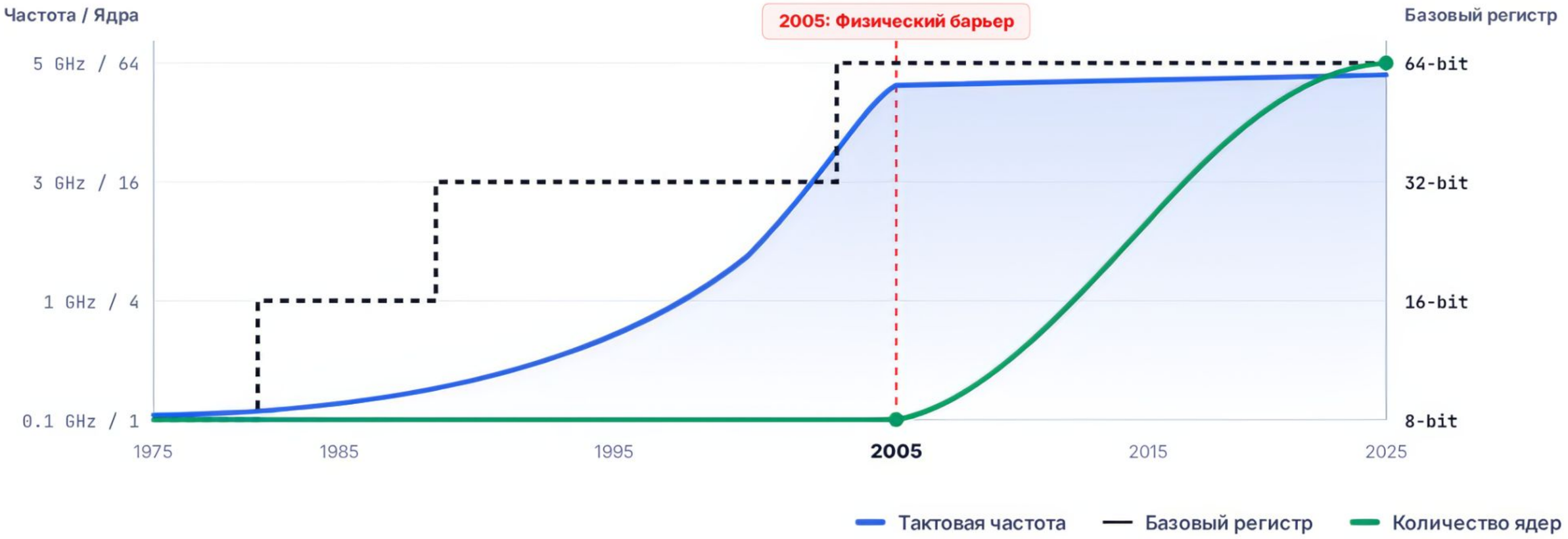
Зачем нам SIMD



План доклада

- → Что такое SIMD?
- → Инструментарий SIMD в .NET
- → Наш эксперимент с SIMD
- → Результаты эксперимента
- → Выводы и советы
- → Полезные ссылки

Эволюция CPU

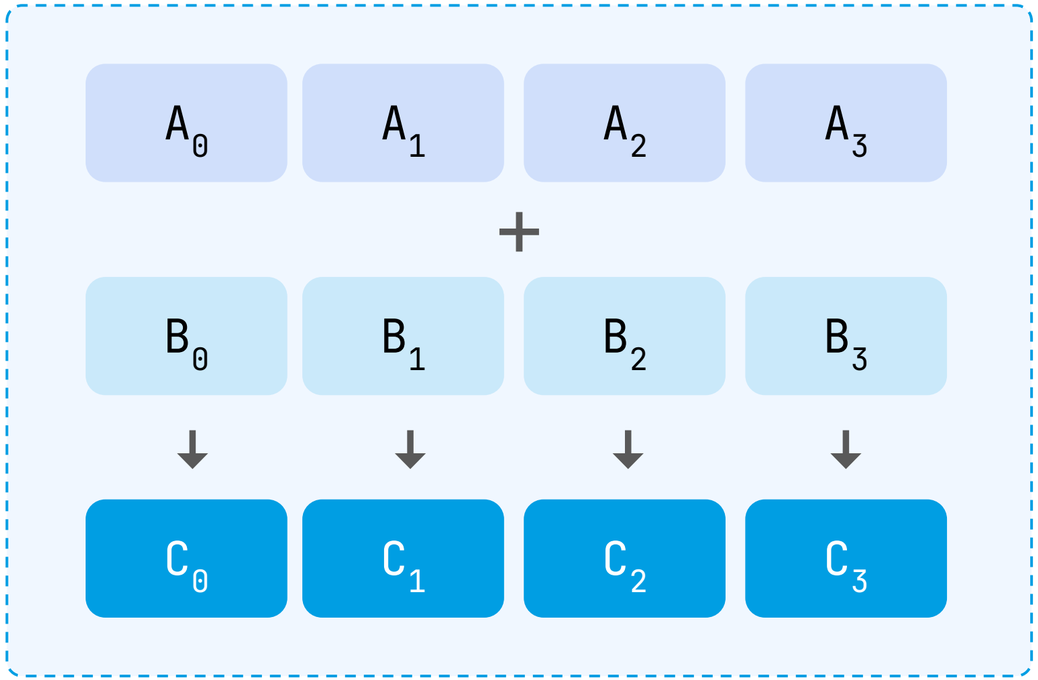


Векторные регистры

Векторные инструкции

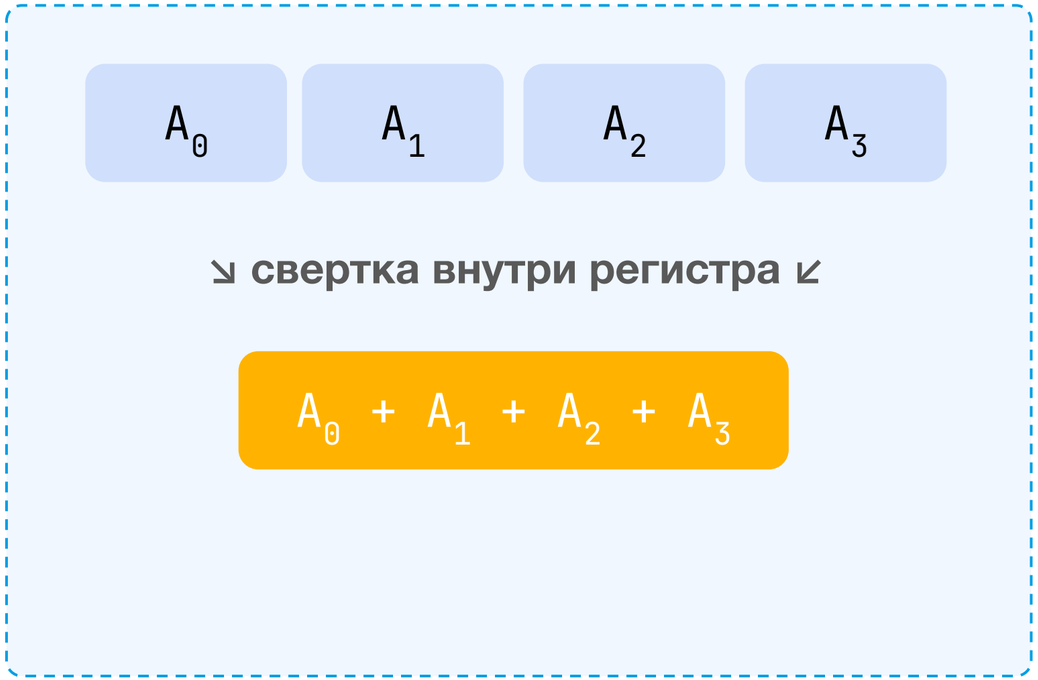
Вертикальные

1 такт



Горизонтальные

3-5 тактов



Стандарт SIMD

	Стандарт	Ширина регистра, бит	Кол-во чисел Float32
x86 / x86-64 Intel, AMD	**SSE**	128	4
	AVX / AVX2	256	8
	AVX-512	512	16
ARM / ARM64 Apple, Qualcomm, AWS	**ARM Neon**	128	4
	SVE / SVE2	от 128 до 2048	от 4 до 64

Где SIMD уже победил?

1 **Колоночные базы данных**

ClickHouse, DuckDB

2 **Локальный интерфейс нейросетей**

llama.cpp, openVINO, ONNIX Runtime

3 **Обработка мультимедиа и графики**

FFmpeg, Skia Graphics Engine, Unity

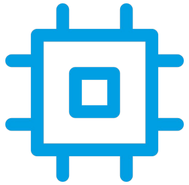
4 **Криптография**

OpenSSL, BoringSSL

Инструментарий .NET

Vector

Абстракция высокого уровня. Адаптируется под железо автоматически



Vector128/256/512

Типизированные векторы фиксированного размера

Intrinsics

Прямой диалог с процессором. Максимальный контроль

Синтетический пример

```
public static double[] Sum(double[] left, double[] right)
{
    int length = left.Length;
    double[] result = new double[length];

   ... Суммирование векторов ...

    return result;
}
```

Vector

```
int vectorSize = Vector<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = new Vector<double>(left, i);
    var v2 = new Vector<double>(right, i);

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Vector

```
int vectorSize = Vector<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = new Vector<double>(left, i);
    var v2 = new Vector<double>(right, i);

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Vector

```
int vectorSize = Vector<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = new Vector<double>(left, i);
    var v2 = new Vector<double>(right, i);

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Vector

```
int vectorSize = Vector<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = new Vector<double>(left, i);
    var v2 = new Vector<double>(right, i);

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Vector

```
int vectorSize = Vector<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = new Vector<double>(left, i);
    var v2 = new Vector<double>(right, i);

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Типизированный Vector

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Типизированный Vector

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Типизированный Vector

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Типизированный Vector

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Типизированный Vector

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    (v1 + v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Hardware Intrinsics

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    Avx2.Add(v1, v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Hardware Intrinsics

```
int vectorSize = Vector256<double>.Count;
int remaining = length % vectorSize;
int i = 0;

for (; i < length - remaining; i += vectorSize)
{
    var v1 = Vector256.Create(left.AsSpan(i));
    var v2 = Vector256.Create(right.AsSpan(i));

    Avx2.Add(v1, v2).CopyTo(result, i);
}

for (; i < length; i++)
{
    result[i] = left[i] + right[i];
}
```

Проверка стандарта

```
Vector.IsHardwareAccelerated // SIMD поддерживается аппаратно

Avx2.IsSupported // Поддержка AVX2

Avx512F.IsSupported // Поддержка AVX-512

AdvSimd.IsSupported // Поддержка ARM Neon

Sve2.IsSupported // Поддержка SVE2
```

Автовекторизация JIT

Идеальные условия

Непрерывная память
Массивы T[ ], Span или ReadOnlySpan

Простой шаг цикла
Строгая инкрементация i++ без прыжков

Понятные границы
Паттерн i < span.Length позволяет JIT убрать проверки **Bounds Check**

Базовая арифметика
Сложение, умножение, сдвиги, Math.Min / Max

Независимость данных
Текущая итерация не зависит от результата предыдущей

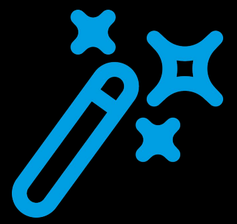
Стоп-факторы

Сложные ветвления
Тяжелые if и фильтры внутри цикла ломают маски

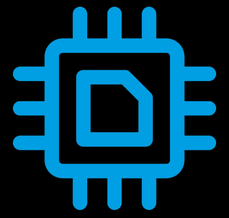
Вызовы методов
Если JIT не смог заинлайнить метод, векторизация отменяется

Приведение типов
Постоянная конвертация (например, int ↔ float) убивает оптимизацию

Общие тренды SIMD в .NET



Умная автовекторизация



Современные стандарты AVX10 и SVE2

Интеграция в прикладной уровень

Фокус на локальные AI-вычисления

Что нового в .NET 11

- Поддержка AVX10 и SVE2
- Поддержка малоразрядных типов
- Изменения в базовой библиотеке (BCL)
- Обновлена модель стоимости инструкций
- Повышение системных требований для ARM64
- Аппаратные маски на x64

Идеальный результат бенчмарка

Method	Median, ns	Ratio
Sum — Scalar	1213.7	1.00
Sum — SIMD	320.9	0.26

Ускорение в 3.7 раза

Вычислительный модуль

SIMD не применим

Расчет матрицы счетов

Сложная логика, ветвления и уникальные алгоритмы для каждого вида спорта.

Идеально для SIMD

Вычисление вероятностей

Суммирования элементов матрицы с фильтрацией по индексам

Последовательный расчет событий (их тысячи):

- **Событие 1:** Победа хозяев — SIMD
- **Событие 2:** Тотал больше 2.5 — SIMD
- …
- **Событие N:** Чётный тотал — SIMD

X4 – X8 **Гипотеза:** ускорение после применения SIMD

Эксперимент

Этап 1

Выделяем фокус-группу

Вместо переписывания всех событий изолируем несколько репрезентативных фильтров

Этап 2

Переписываем код

Реализуем логику обработки только для фокус-группы в новом векторном формате SIMD

~~for~~

↓

Vector

Этап 3

Сравниваем

Проводим замеры производительности изолированной части

Наша цель

Проверить **жизнеспособность идеи** на нескольких событиях, прежде чем трогать остальной код

Скалярный код

```
public static Probability CalculateProbability(double[][] matrix)
{
    var resultProbability = 0d;

    for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
    {
        var minAway = int.IsEvenInteger(homeScore) ? 0 : 1;

        for (var awayScore = minAway; awayScore < matrix[homeScore].Length; awayScore += 2)
        {
            resultProbability += matrix[homeScore][awayScore];
        }
    }

    return new(resultProbability);
}
```

Скалярный код

```
public static Probability CalculateProbability(double[][] matrix)
{
    var resultProbability = 0d;

    for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
    {
        var minAway = int.IsEvenInteger(homeScore) ? 0 : 1;

        for (var awayScore = minAway; awayScore < matrix[homeScore].Length; awayScore += 2)
        {
            resultProbability += matrix[homeScore][awayScore];
        }
    }

    return new(resultProbability);
}
```

Скалярный код

```
public static Probability CalculateProbability(double[][] matrix)
{
    var resultProbability = 0d;

    for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
    {
        var minAway = int.IsEvenInteger(homeScore) ? 0 : 1;

        for (var awayScore = minAway; awayScore < matrix[homeScore].Length; awayScore += 2)
        {
            resultProbability += matrix[homeScore][awayScore];
        }
    }

    return new(resultProbability);
}
```

Скалярный код

```
public static Probability CalculateProbability(double[][] matrix)
{
    var resultProbability = 0d;

    for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
    {
        var minAway = int.IsEvenInteger(homeScore) ? 0 : 1;

        for (var awayScore = minAway; awayScore < matrix[homeScore].Length; awayScore += 2)
        {
            resultProbability += matrix[homeScore][awayScore];
        }
    }

    return new(resultProbability);
}
```

Векторный код

```
var vectorSize = Vector<double>.Count;
var sumVector = Vector<double>.Zero;

var evenIndicesMask = new double[vectorSize];
var oddIndicesMask = new double[vectorSize];

for (var i = 0; i < vectorSize; i++)
{
   evenIndicesMask[i] = int.IsEvenInteger(i) ? 1d : 0d;
   oddIndicesMask[i]  = int.IsEvenInteger(i) ? 0d : 1d;
}

var evenMultiplier = new Vector<double>(evenIndicesMask);
var oddMultiplier = new Vector<double>(oddIndicesMask);

... Вычисление ...
```

Векторный код

```
var vectorSize = Vector<double>.Count;
var sumVector = Vector<double>.Zero;

var evenIndicesMask = new double[vectorSize];
var oddIndicesMask = new double[vectorSize];

for (var i = 0; i < vectorSize; i++)
{
   evenIndicesMask[i] = int.IsEvenInteger(i) ? 1d : 0d;
   oddIndicesMask[i]  = int.IsEvenInteger(i) ? 0d : 1d;
}

var evenMultiplier = new Vector<double>(evenIndicesMask);
var oddMultiplier = new Vector<double>(oddIndicesMask);

... Вычисление ...
```

Векторный код

```
var vectorSize = Vector<double>.Count;
var sumVector = Vector<double>.Zero;

var evenIndicesMask = new double[vectorSize];
var oddIndicesMask = new double[vectorSize];

for (var i = 0; i < vectorSize; i++)
{
   evenIndicesMask[i] = int.IsEvenInteger(i) ? 1d : 0d;
   oddIndicesMask[i]  = int.IsEvenInteger(i) ? 0d : 1d;
}

var evenMultiplier = new Vector<double>(evenIndicesMask);
var oddMultiplier = new Vector<double>(oddIndicesMask);

... Вычисление ...
```

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    for (; awayScore < matrix[homeScore].Length ; awayScore += vectorSize)
    {
        var data = new Vector<double>(matrix[homeScore], awayScore);

        var multiplier = int.IsEvenInteger(homeScore + awayScore)
            ? evenMultiplier
            : oddMultiplier;

        sumVector += data * multiplier;
    }

    ... Обработка оставшийся элементов ...
}

resultProbability += Vector.Sum(sumVector);
```

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    for (; awayScore < matrix[homeScore].Length ; awayScore += vectorSize)
    {
        var data = new Vector<double>(matrix[homeScore], awayScore);

        var multiplier = int.IsEvenInteger(homeScore + awayScore)
            ? evenMultiplier
            : oddMultiplier;

        sumVector += data * multiplier;
    }

    ... Обработка оставшийся элементов ...
}

resultProbability += Vector.Sum(sumVector);
```

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    for (; awayScore < matrix[homeScore].Length ; awayScore += vectorSize)
    {
        var data = new Vector<double>(matrix[homeScore], awayScore);

        var multiplier = int.IsEvenInteger(homeScore + awayScore)
            ? evenMultiplier
            : oddMultiplier;

        sumVector += data * multiplier;
    }

    ... Обработка оставшийся элементов ...
}

resultProbability += Vector.Sum(sumVector);
```

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    for (; awayScore < matrix[homeScore].Length ; awayScore += vectorSize)
    {
        var data = new Vector<double>(matrix[homeScore], awayScore);

        var multiplier = int.IsEvenInteger(homeScore + awayScore)
            ? evenMultiplier
            : oddMultiplier;

        sumVector += data * multiplier;
    }

    ... Обработка оставшийся элементов ...
}

resultProbability += Vector.Sum(sumVector);
```

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    ... Обработка элементов векторами …

    for (; awayScore < matrix[homeScore].Length; awayScore++)
    {
        if (int.IsEvenInteger(homeScore + awayScore))
        {
            resultProbability += matrix[homeScore][awayScore];
        }
    }
}

var resultProbability += Vector.Sum(sumVector);
```

Динамическое выравнивание строк

Обычный массив (размер 11)

⚠ Медленный скалярный «хвост»

С динамическим padding (размер 12)

✓ Только векторная обработка

Vector Scalar Padding

Векторный код

```
for (var homeScore = 0; homeScore < matrix.Length; homeScore++)
{
    var awayScore = 0;

    for (; awayScore < matrix[homeScore].Length ; awayScore += vectorSize)
    {
        var data = new Vector<double>(matrix[homeScore], awayScore);

        var multiplier = int.IsEvenInteger(homeScore + awayScore)
            ? evenMultiplier
            : oddMultiplier;

        sumVector += data * multiplier;
    }
}

resultProbability = Vector.Sum(sumVector);
```

Сравнение

Method	Median, us	Ratio
Calculate — Scalar	2.816	1.00
Calculate — SIMD	2.349	0.83

Ускорения почти нет

Может быть автовекторизация?

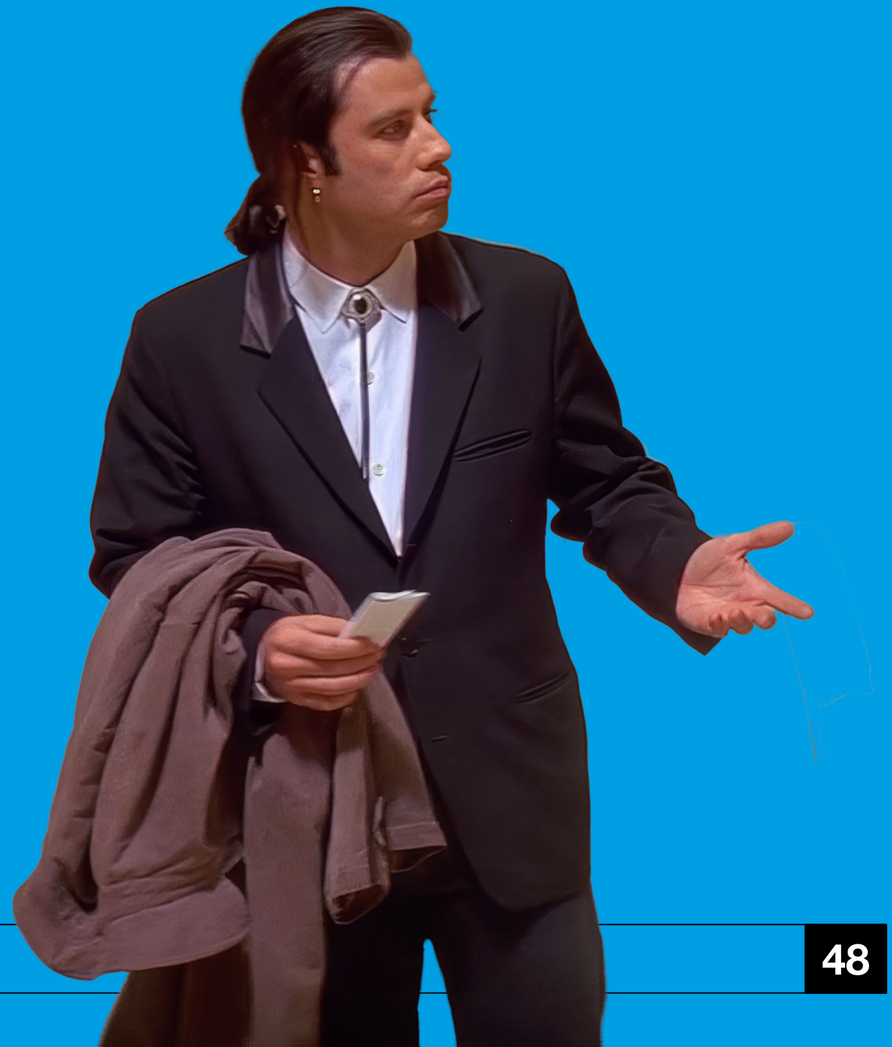
ФАКТИЧЕСКИЙ JIT-КОД (SCALAR)

```
M01_L00:
     ...
     mov        r10,[rbx+r8*8+10]

     vaddsd     xmm0,xmm0,qword ptr
[r10+r8*8+10]
     ; Инструкция sd (Scalar Double)
     ; Минимальный регистр xmm0 (128 бит)
     ; Обрабатываем по одному числу за раз

     inc        edx
     ; inc увеличивает шаг цикла на 1

     cmp        edx,eax
     jl         short M01_L00
     ; Прыжок на начало скалярного цикла
```

ГИПОТЕТИЧЕСКИЙ КОД (SIMD)

```
M01_L35:
     ...
     mov        r10d,edx

     vaddpd     ymm1,ymm1,[r8+r10*8+10]
     ; Инструкция pd (Packed Double)
     ; Регистр ymm1 (256 бит).
     ; За один раз складываются 4 элемента

     add        edx,4
     ; add увеличивает шаг цикла на 4

     cmp        r8d,edx
     jge        short M01_L35
     ; Прыжок на начало векторного цикла
```

Что дальше?

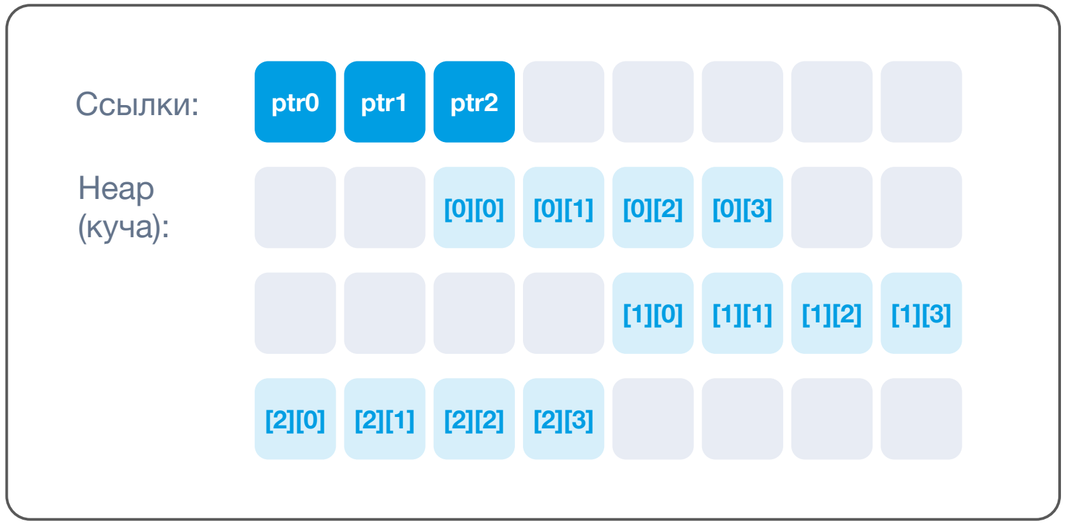


Оптимизация 1: Линеаризация матриц

Высокая фрагментация

Зубчатый массив (Jagged)

```
double[][] _matrix;
```

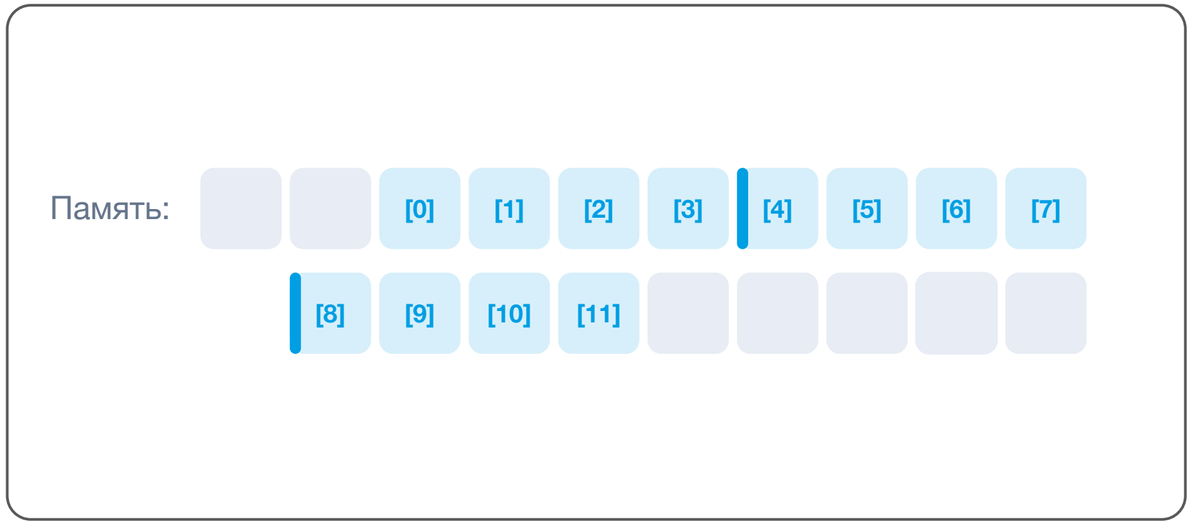


Кэш-лояльность

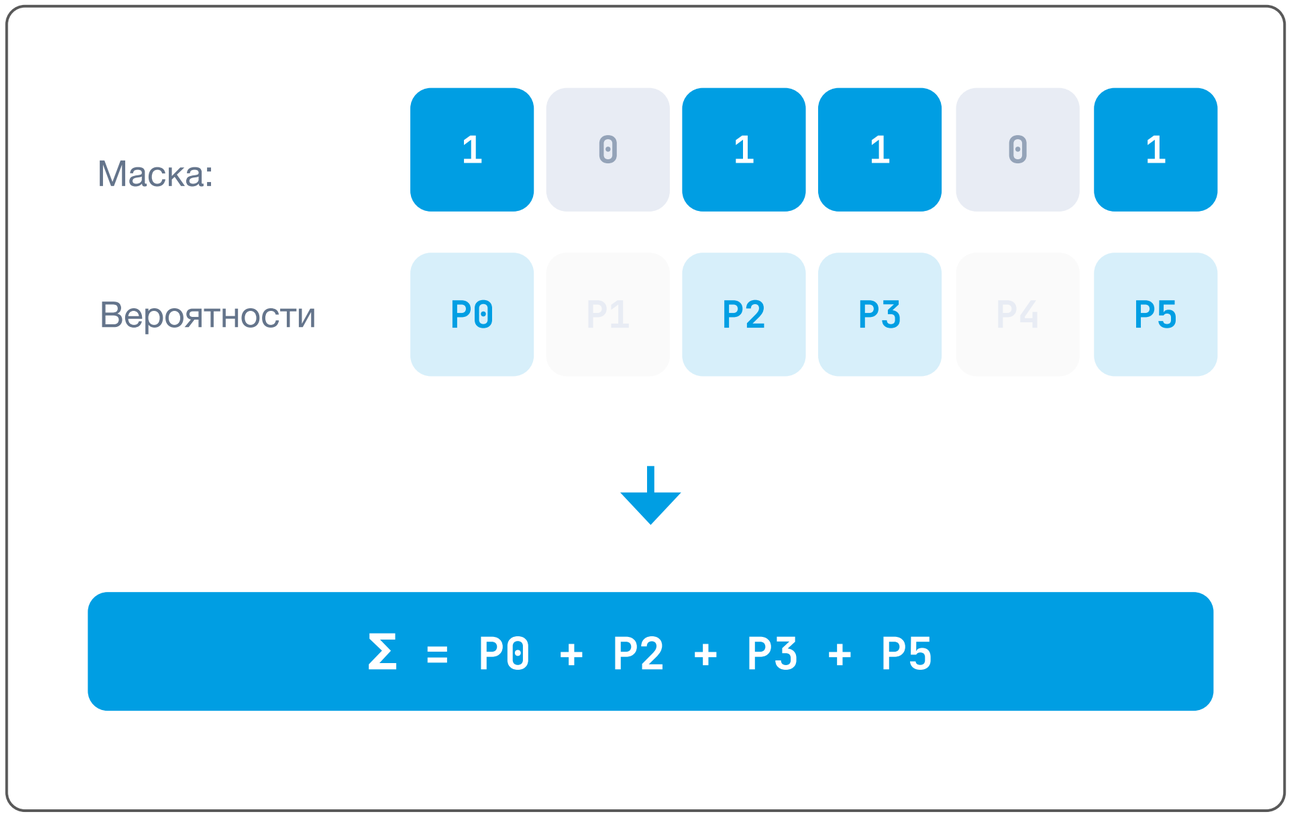
Плоский массив (Flat)

```
double[] _flatfMatrix;
```

```
index = row * width + col
```



Оптимизация 2: Векторные маски вместо ветвлений



Векторные маски

Генерируется специальная векторная маска. С помощью аппаратного селектора отфильтрованные данные готовы к суммированию.

```
Vector<long> mask = Vector.GreaterThan(
    vector1,
    vector2);

var result = Vector.ConditionalSelect(
    mask,
    vector,
    Vector<double>.Zero);
```

Финальный код

```
public static Probability CalculateProbability(ReadOnlySpan<double> flatMatrix, int size)
{
    var vectorSize = Vector<double>.Count;
    var sumVector = Vector<double>.Zero;

    var sizeVector = new Vector<long>(size);
    var stepVector = new Vector<long>(vectorSize);
    var oneVector = Vector<long>.One;

    var initialAwayScores = new long[vectorSize];
    for (var i = 0; i < vectorSize; i++)
    {
        initialAwayScores[i] = i;
    }

    var homeScores = new Vector<long>(new long[vectorSize]);
    var awayScores = new Vector<long>(initialAwayScores);

    ... Вычисление …

    return new(resultProbability);
}
```

Финальный код

```
public static Probability CalculateProbability(ReadOnlySpan<double> flatMatrix, int size)
{
    var vectorSize = Vector<double>.Count;
    var sumVector = Vector<double>.Zero;

    var sizeVector = new Vector<long>(size);
    var stepVector = new Vector<long>(vectorSize);
    var oneVector = Vector<long>.One;

    var initialAwayScores = new long[vectorSize];
    for (var i = 0; i < vectorSize; i++)
    {
        initialAwayScores[i] = i;
    }

    var homeScores = new Vector<long>(new long[vectorSize]);
    var awayScores = new Vector<long>(initialAwayScores);

    ... Вычисление …

    return new(resultProbability);
}
```

Финальный код

```
public static Probability CalculateProbability(ReadOnlySpan<double> flatMatrix, int size)
{
    var vectorSize = Vector<double>.Count;
    var sumVector = Vector<double>.Zero;

    var sizeVector = new Vector<long>(size);
    var stepVector = new Vector<long>(vectorSize);
    var oneVector = Vector<long>.One;

    var initialAwayScores = new long[vectorSize];
    for (var i = 0; i < vectorSize; i++)
    {
        initialAwayScores[i] = i;
    }

    var homeScores = new Vector<long>(new long[vectorSize]);
    var awayScores = new Vector<long>(initialAwayScores);

    ... Вычисление …

    return new(resultProbability);
}
```

Финальный код

```
public static Probability CalculateProbability(ReadOnlySpan<double> flatMatrix, int size)
{
    var vectorSize = Vector<double>.Count;
    var sumVector = Vector<double>.Zero;

    var sizeVector = new Vector<long>(size);
    var stepVector = new Vector<long>(vectorSize);
    var oneVector = Vector<long>.One;

    var initialAwayScores = new long[vectorSize];
    for (var i = 0; i < vectorSize; i++)
    {
        initialAwayScores[i] = i;
    }

    var homeScores = new Vector<long>(new long[vectorSize]);
    var awayScores = new Vector<long>(initialAwayScores);

    ... Вычисление …

    return new(resultProbability);
}
```

Финальный код

```
for (var i = 0; i < flatMatrix.Length; i += vectorSize)
{
    var data = new Vector<double>(flatMatrix.Slice(i));

    var totalSum = homeScores + awayScores;

    var mask = Vector.IsEvenInteger(totalSum);

    sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

    var newAwayScores = awayScores + stepVector;
    var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

    awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
    homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
}

var resultProbability = Vector.Sum(sumVector);
```

Финальный код

```
for (var i = 0; i < flatMatrix.Length; i += vectorSize)
{
    var data = new Vector<double>(flatMatrix.Slice(i));

    var totalSum = homeScores + awayScores;

    var mask = Vector.IsEvenInteger(totalSum);

    sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

    var newAwayScores = awayScores + stepVector;
    var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

    awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
    homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
}

var resultProbability = Vector.Sum(sumVector);
```

Финальный код

```
for (var i = 0; i < flatMatrix.Length; i += vectorSize)
{
    var data = new Vector<double>(flatMatrix.Slice(i));

    var totalSum = homeScores + awayScores;

    var mask = Vector.IsEvenInteger(totalSum);

    sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

    var newAwayScores = awayScores + stepVector;
    var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

    awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
    homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
}

var resultProbability = Vector.Sum(sumVector);
```

Финальный код

```
for (var i = 0; i < flatMatrix.Length; i += vectorSize)
{
    var data = new Vector<double>(flatMatrix.Slice(i));

    var totalSum = homeScores + awayScores;

    var mask = Vector.IsEvenInteger(totalSum);

    sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

    var newAwayScores = awayScores + stepVector;
    var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

    awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
    homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
}

var resultProbability = Vector.Sum(sumVector);
```

Сравнение

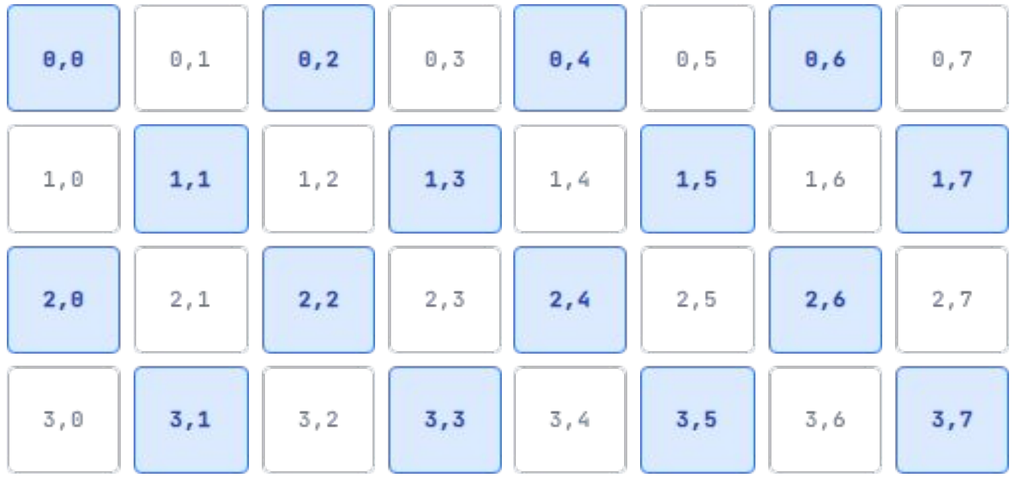
Method	Median, us	Ratio
Calculate — Scalar	2.816	1.00
Calculate — SIMD	2.349	0.83
Calculate — SIMD Optimized	5.195	1.84

Стало медленнее

Почему?

Пример первый

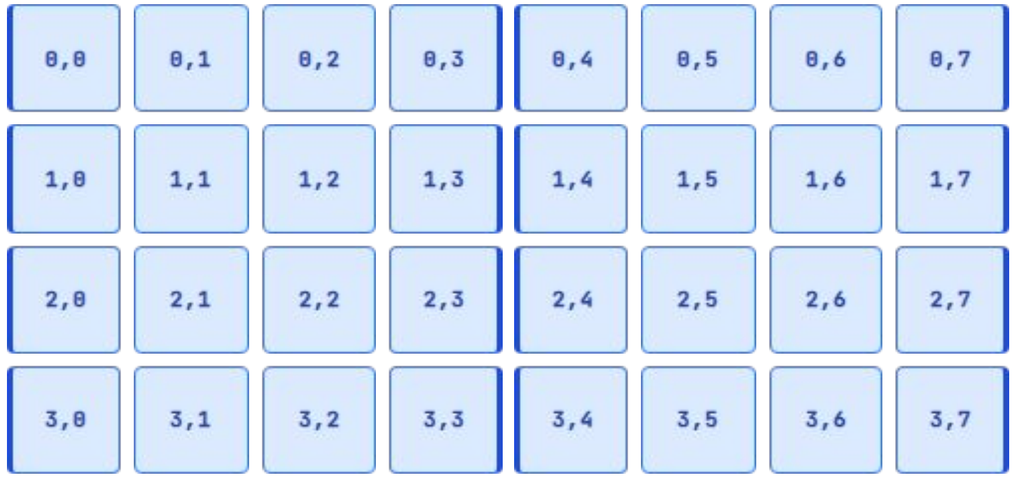
Скалярный проход



$$\text{Операции}_{\text{скаляр}} = \frac{n \times m}{2}$$

Условная проверка и обработка

Векторный проход (SIMD)



$$\text{Операции}_{\text{вектор}} = \frac{n \times m}{4}$$

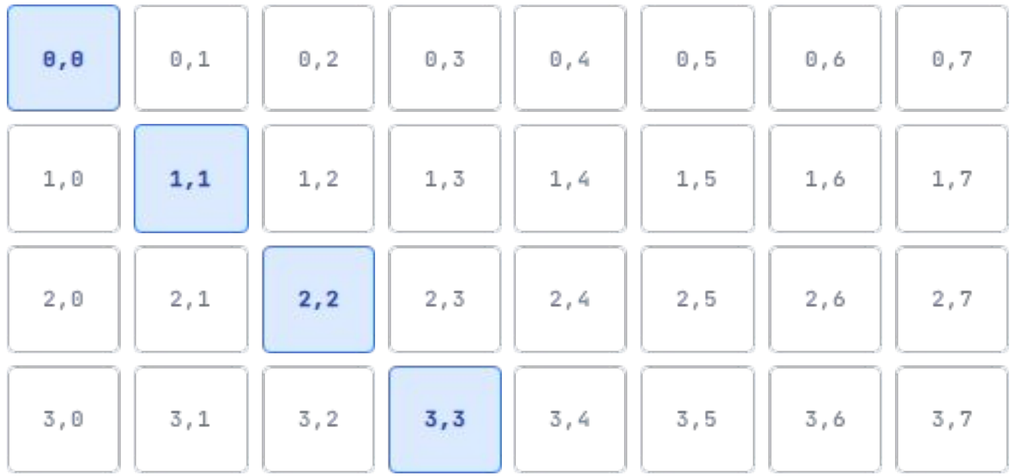
Загрузка блоков по 4 элемента

Пример первый. Сравнение

Method	Median, ns	Ratio
Calculate Even — Scalar	478.3	1.00
Calculate Even — SIMD	196.6	0.41

Пример второй

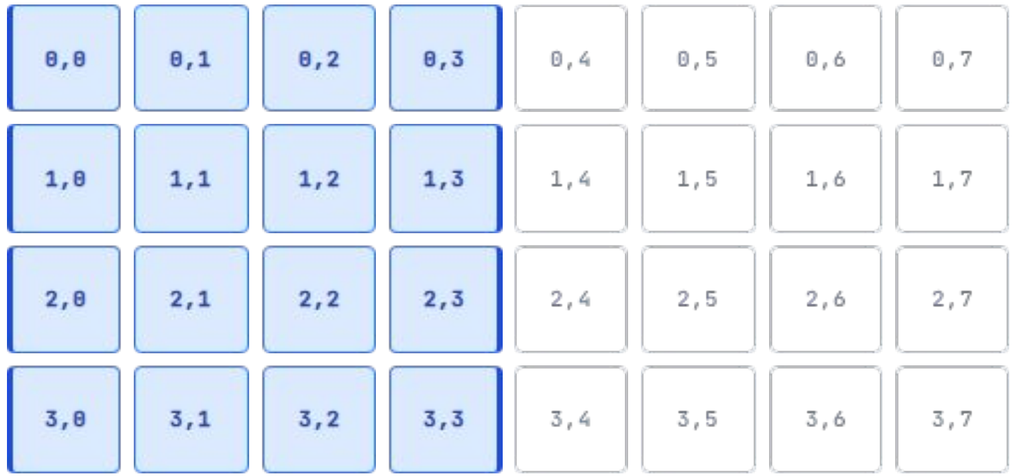
Скалярный проход



$$Операции_{скаляр} = \min(n, m)$$

Обработка диагональных элементов

Векторный проход (SIMD)



$$Операции_{вектор} = \min(n, m)$$

Обработка векторов с диагональными элементами

Пример второй. Сравнение

Method	Median, ns	Ratio
Calculate Draw — Scalar	20.65	1.00
Calculate Draw — SIMD	91.89	4.45

Векторизация с масками

```
var data = new Vector<double>(flatMatrix.Slice(i));

var totalSum = homeScores + awayScores;

var mask = Vector.IsEvenInteger(totalSum);

sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

var newAwayScores = awayScores + stepVector;
var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
```

Векторизация с масками

```
var data = new Vector<double>(flatMatrix.Slice(i));

var totalSum = homeScores + awayScores;

var mask = Vector.IsEvenInteger(totalSum);

sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

var newAwayScores = awayScores + stepVector;
var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
```

Векторизация с масками

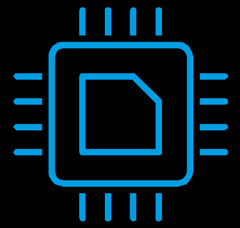
```
var data = new Vector<double>(flatMatrix.Slice(i));

var totalSum = homeScores + awayScores;

var mask = Vector.IsEvenInteger(totalSum);

sumVector += Vector.ConditionalSelect(mask, data, Vector<double>.Zero);

var newAwayScores = awayScores + stepVector;
var wrapMask = Vector.GreaterThanOrEqual(newAwayScores, sizeVector);

awayScores = Vector.ConditionalSelect(wrapMask, newAwayScores - sizeVector, newAwayScores);
homeScores = Vector.ConditionalSelect(wrapMask, homeScores + oneVector, awayScores);
```

Пример первый. Сравнение

Method	Median, ns	Ratio
Calculate Even — Scalar	478.3	1.00
Calculate Even — SIMD	196.6	0.41
Calculate Even — SIMD Optimized	537.7	1.12

Физические ограничения

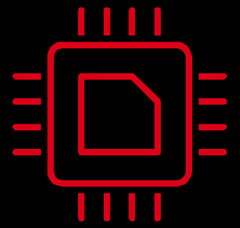
Normal Operation



4.5 GHz

`Scalar / Integer Code`

Heavy SIMD Load



3.8 GHz

`256/512-bit AVX Units`

AVX Frequency Offset / Thermal Throttling

Итог эксперимента

Method	Median, us	Ratio
Calculate — Scalar	2.531	1.00
Calculate — SIMD	0.976	0.39
Calculate — SIMD Optimized	3.453	1.36

- Эффект на всех событиях не ясен
- Сложность кода возрастает кратно
- Нужна поддержка двух подходов

Как решилась бизнес-задача?

- Оптимизация работы с памятью и снятие нагрузки с GC
- Точечные алгоритмические улучшения

Когда SIMD не нужен

- I/O bound сценарии
- Параллелизм — ловушка падения частоты
- Сложная логика подготовки данных
- Высокая цена поддержки

Правила игры

- Сначала проектируйте расположение данных в памяти
- Никакой веры на слово — только честные замеры

Полезные материалы

.NET & Hardware Intrinsics

- Hardware Intrinsics in .NET Core 3.0 and beyond
- Документация по SIMD и Hardware Intrinsics в .NET
- Справочник API: Пространство имен System.Numerics
- Справочник API: System.Runtime.Intrinsics.X86 / Arm

Архитектура процессоров и инструкции

- Intel® Intrinsics Guide
- ARM Architecture Reference Manual
- Agner Fog’s Instruction Tables & Microarchitecture Guides

Спасибо
за внимание!

Кирилл Панин

Старший .NET разработчик Altenar

@kapanin590