

АНАЛИЗ И ОПТИМИЗАЦИЯ ПРИМЕНЕНИЯ ВЕКТОРНЫХ SIMD-ИНСТРУКЦИЙ ДЛЯ ВЕКТОРИЗАЦИИ КОДА

Выполнил: Буглин Андрей Павлович, гр. 9305

Руководитель: Пазников Алексей Александрович, к.т.н., доцент



Санкт-Петербург | 2023

СПбГЭТУ "ЛЭТИ"



ЦЕЛЬ И ЗАДАЧИ

ЦЕЛЬ

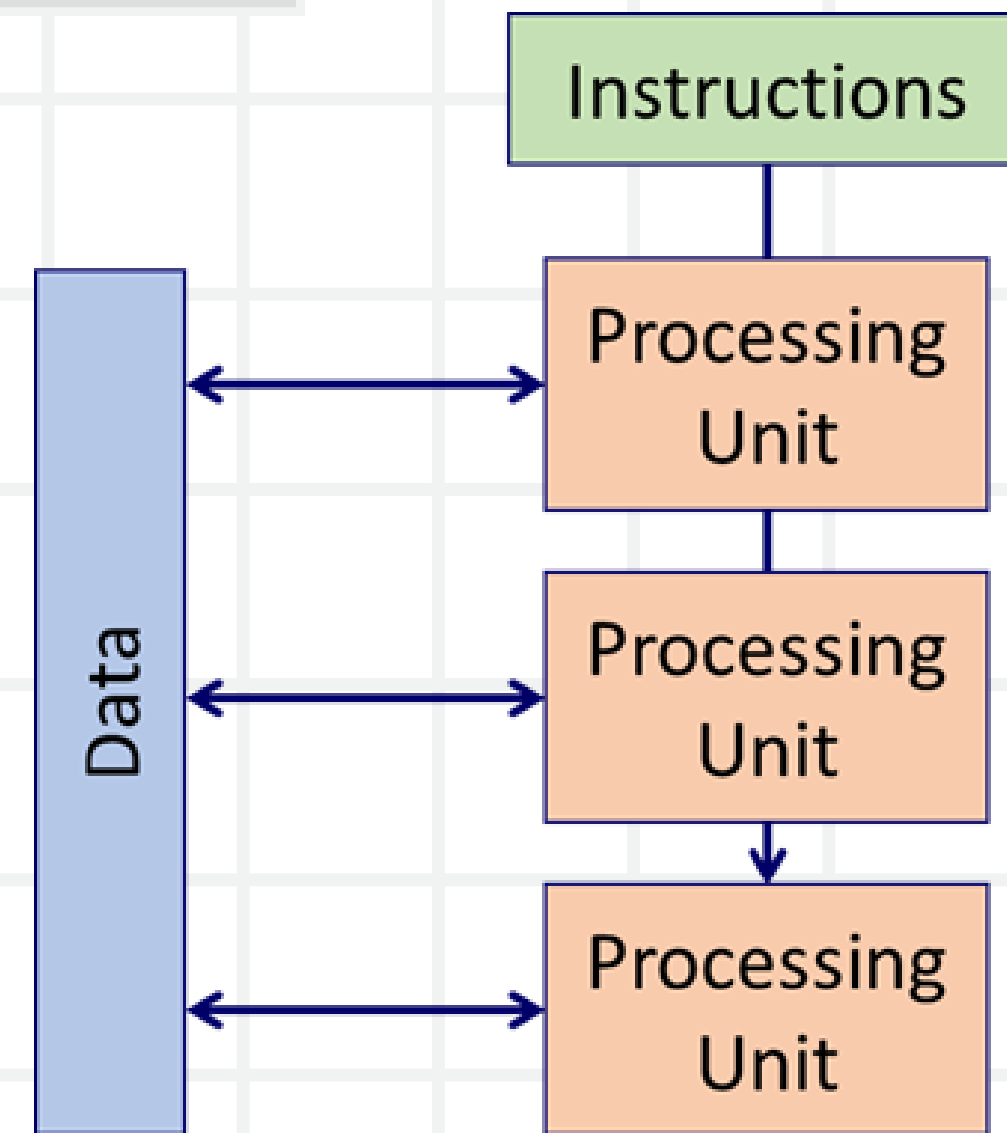
Изучение и анализ векторных SIMD-инструкций с целью разработки оптимизированных алгоритмов, которые могут эффективно использовать эти инструкции.


ЗАДАЧИ

- Изучить принципы работы векторных SIMD-инструкций.
- Разработать оптимизированные версии алгоритмов, максимально использующие возможности векторных SIMD-инструкций.
- Оценить полученные результаты и сделать выводы о применимости и эффективности использования векторных SIMD-инструкций.

SINGLE INSTRUCTION, MULTIPLE DATA (SIMD)

- Векторизация кода – техника оптимизации вычислений, которая позволяет выполнить однотипные операции над набором данных одновременно.
- Цель векторизации кода – увеличение производительности
- Применение векторизации кода позволяет обрабатывать большие объемы данных параллельно и ускорить выполнение алгоритмов.



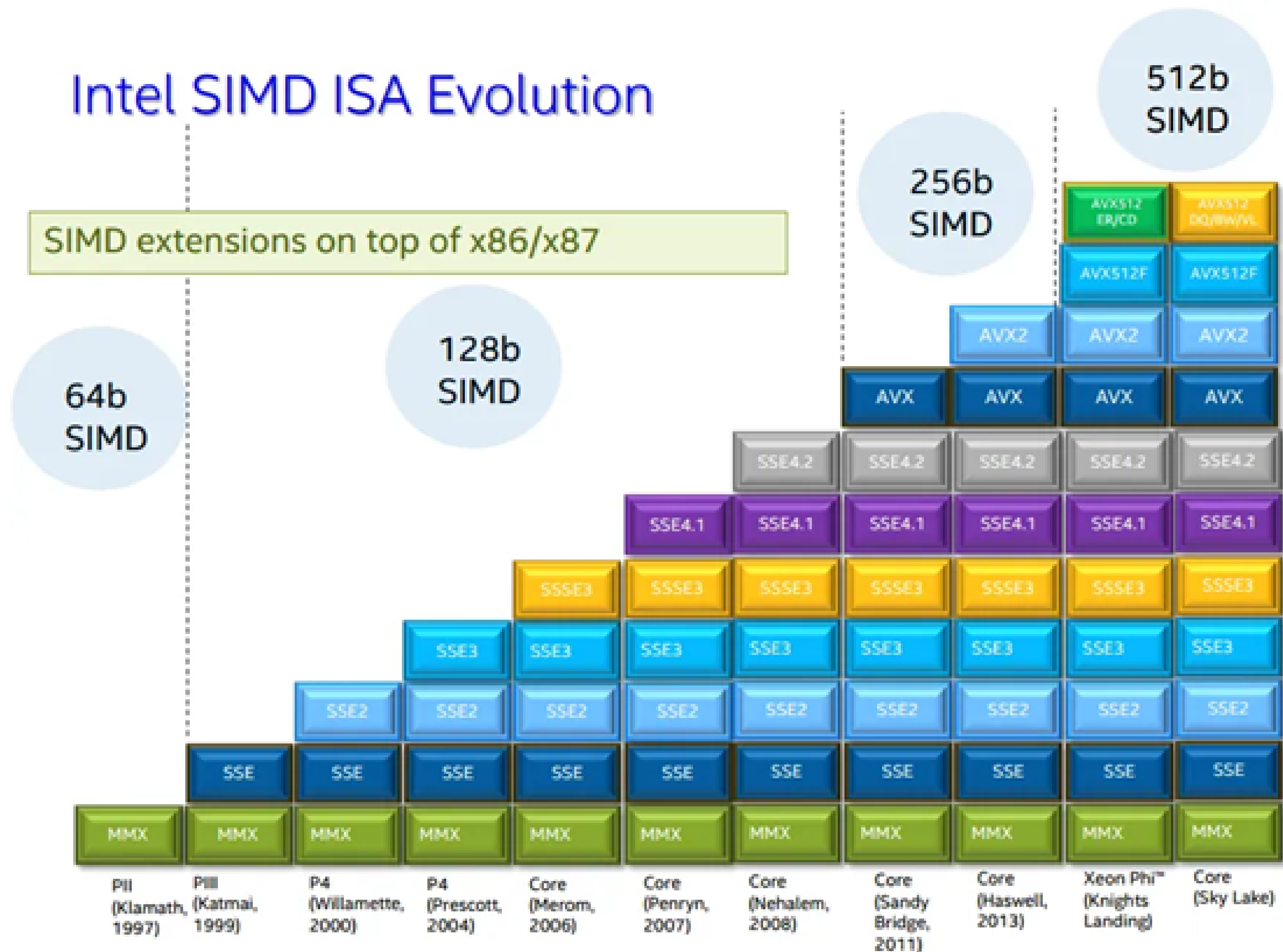


РАСШИРЕНИЯ INTEL SIMD

- Новые инструкции, новые регистры
- Вводится поэтапно/группами функциональных возможностей
- SSE – SSE4 операции с шириной 128 бит
- AVX, FMA, AVX2, AVX-512 операции с шириной от 256 до 512 бит

Intel SIMD ISA Evolution

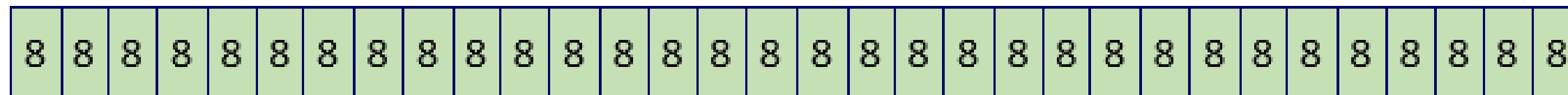
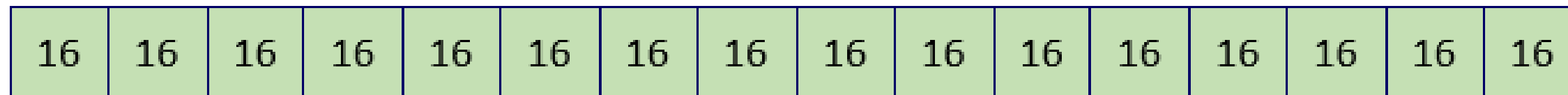
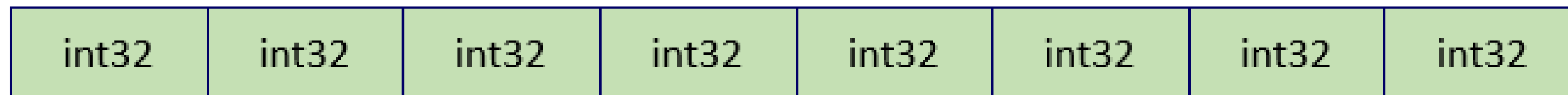
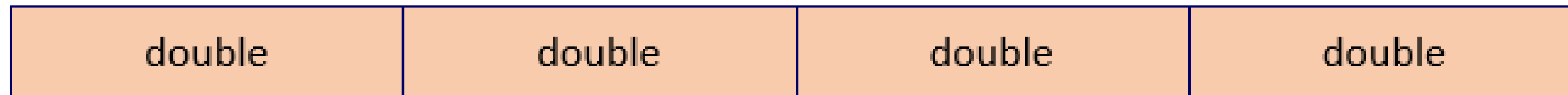
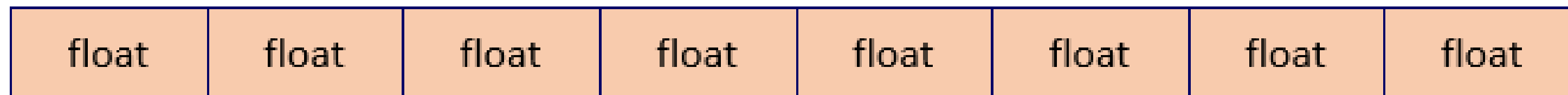
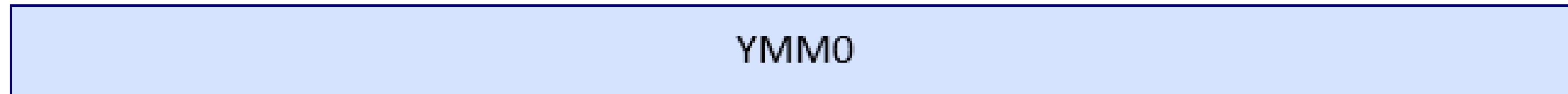
SIMD extensions on top of x86/x87



ТИПЫ ДАННЫХ SSE/AVX

255

0



Операция с 32
8-битными
значениями в
одной
инструкции!

АВТОМАТИЧЕСКАЯ ВЕКТОРИЗАЦИЯ

- Большинство компиляторов имеют некоторый уровень поддержки автоматической векторизации / SIMD
- Хотя это полезно, нельзя гарантировать, что оно будет работать постоянно.
- Небольшие изменения в коде могут привести к большим проблемам с изменением способа генерации кода

```
int a[256], b[256], c[256];  
void foo () {  
    for (int i=0; i<256; i++) a[i] = b[i] * c[i];  
}
```

ВЫБРАННЫЕ АЛГОРИТМЫ

СРЕДНЕЕ ЗНАЧЕНИЕ ВЕКТОРА

Нахождение
суммы элементов
деленное на
количество

УМНОЖЕНИЕ МАТРИЦЫ НА ВЕКТОР

Скалярного
произведение
соответствующей
строки матрицы и
вектора

ТРАНСПОНИРОВАНИЕ МАТРИЦЫ

Строки матрицы
становятся
столбцами и
наоборот

НАХОЖДЕНИЕ ОБРАТНОЙ МАТРИЦЫ

Преобразует
исходную матрицу,
при умножении на
которую получается
единичная матрица

ПРЕОБРАЗОВАНИЕ ВЕРШИНЫ

Координаты вершины
умножаются на
соответствующие
элементы матрицы
преобразования.

ВЫЧИСЛЕНИЕ СРЕДНЕГО ЗНАЧЕНИЯ ВЕКТОРА



```
53 static float SimdAverageKernel()
54 {
55     preventOptimize++;
56     __m128 sumx4 = _mm_set_ps1(0.0);
57     for (uint32_t j = 0, l = length; j < l; j = j + 4)
58     {
59         sumx4 = _mm_add_ps(sumx4, _mm_loadu_ps(&a[j]));
60     }
61     Base::Lanes<__m128, float> lanes(sumx4);
62     return (lanes.x() + lanes.y() + lanes.z() + lanes.w()) / length;
63 }
```



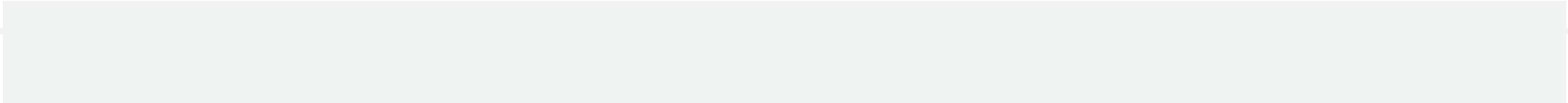
СРАВНИТЕЛЬНЫЙ АНАЛИЗ

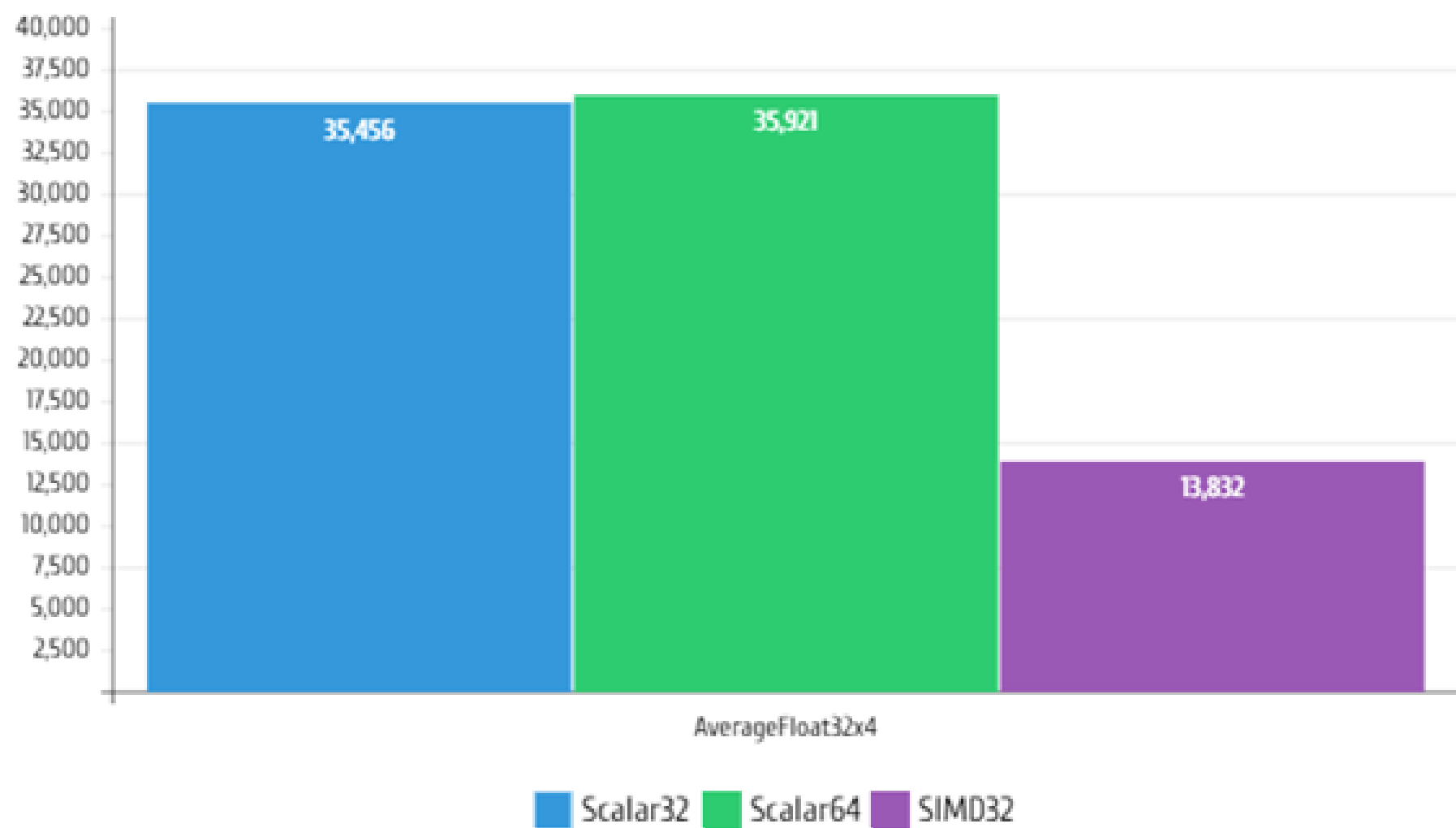
- Выбор алгоритмов
- Программная реализация
- Используемые инструменты и средства измерения.
Профилирование
- Характеристики системы
- Входные данные
- Методика измерения



РЕЗУЛЬТАТЫ ОПТИМИЗАЦИИ

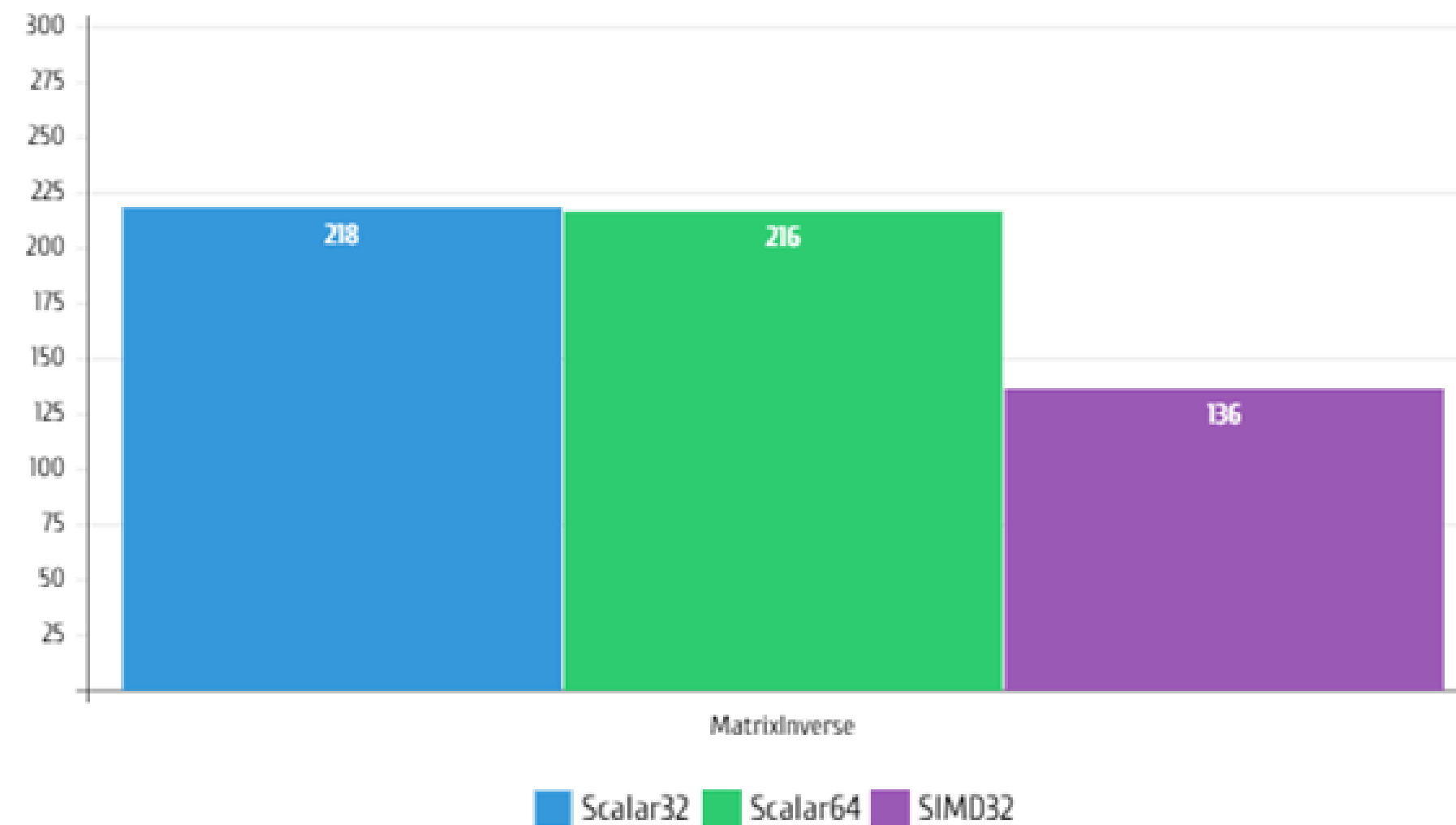
Name	:	Iterations	Scalar32(ns)	Scalar64(ns)	SIMD32(ns)
AverageFloat32x4	:	27978	35456	35921	13832
VertexTransform	:	1424695	701	262	61
MatrixMultiplication	:	18267874	53	93	52
MatrixTranspose	:	49344752	20	14	14
MatrixInverse	:	4615211	218	216	136





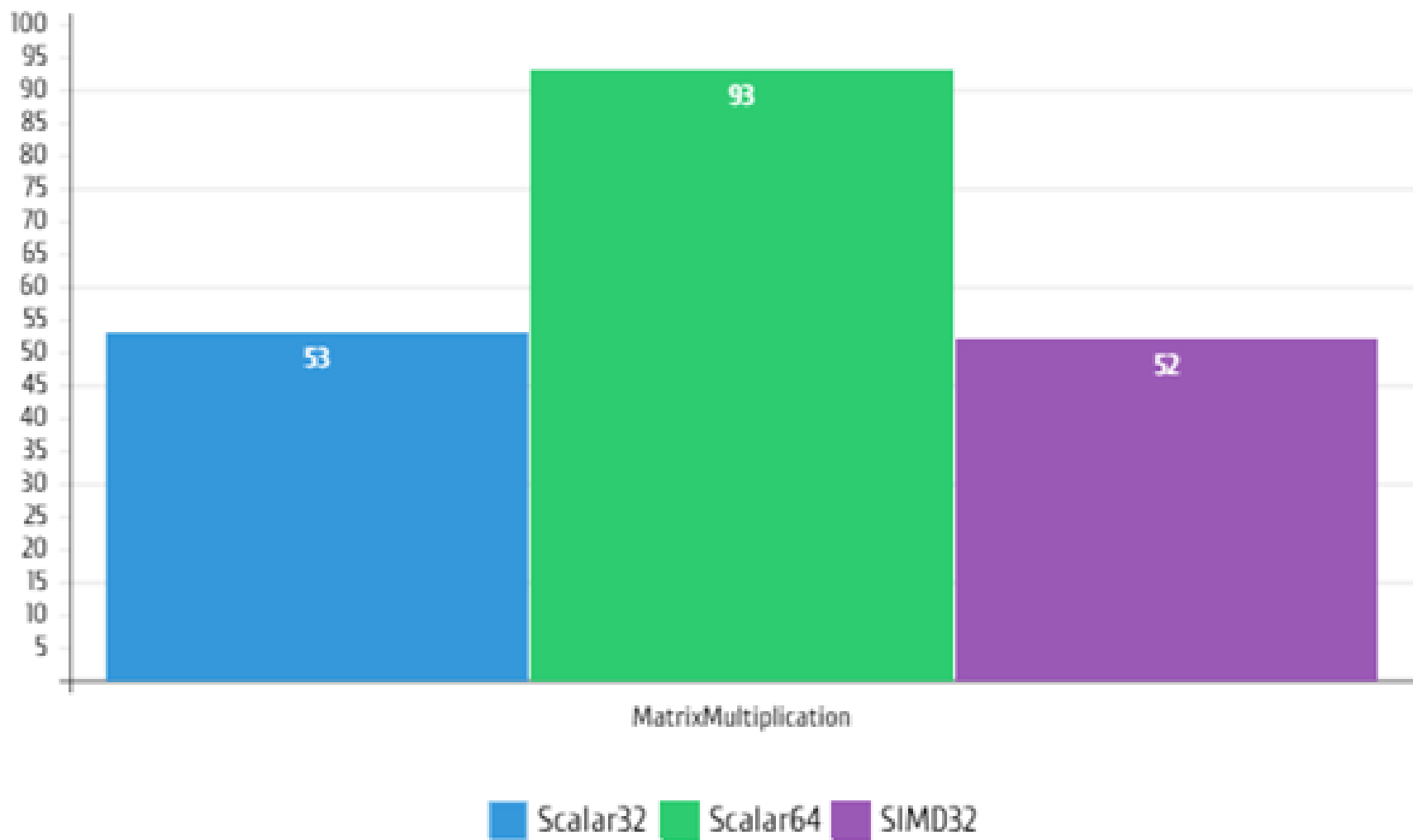
Алгоритм "Вычисление среднего значения вектора"

- Scalar 32: 35456 наносекунд.
- SIMD-инструкций: 13832 наносекунд
- Ускорение в 2.56 раз



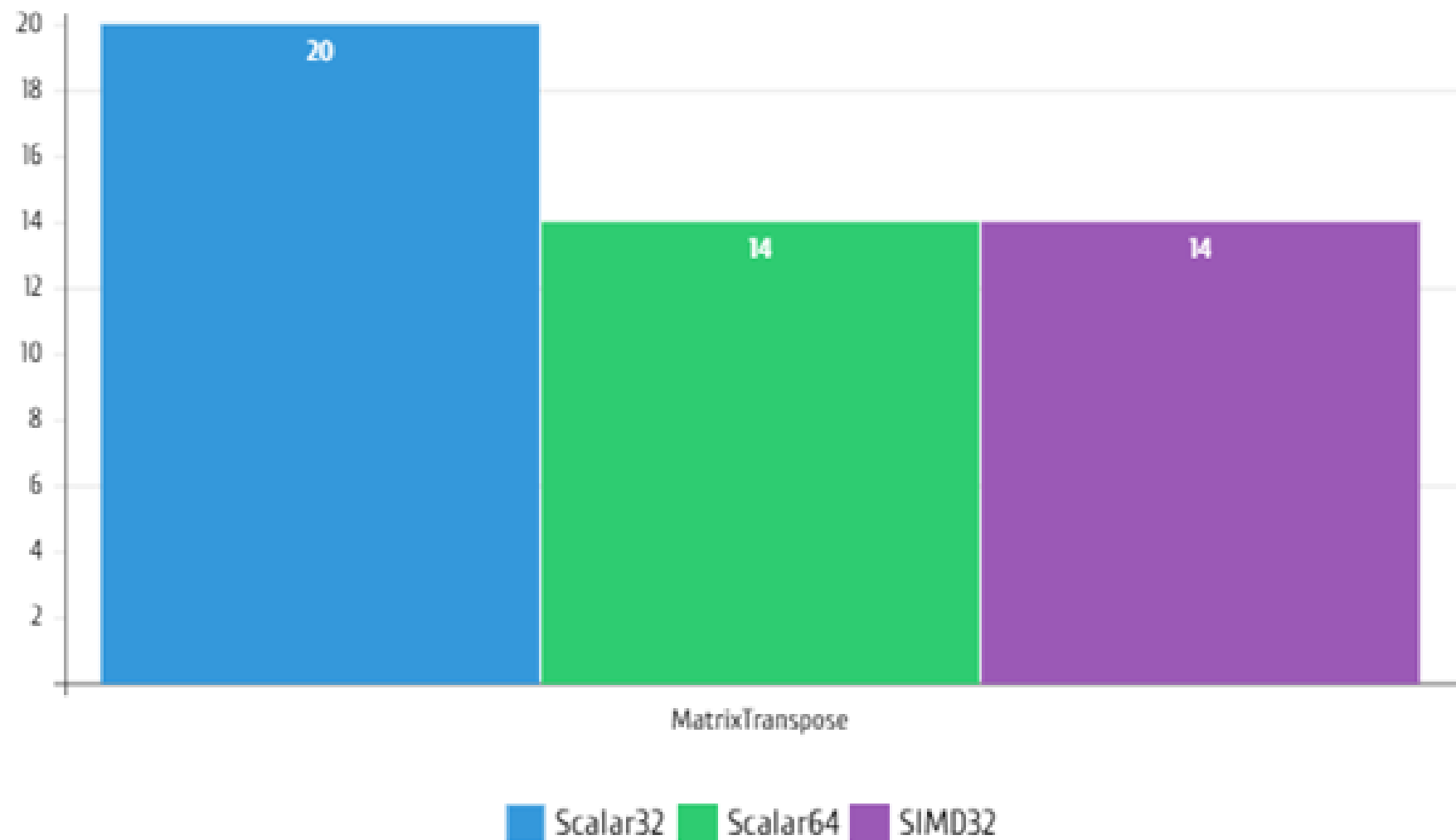
Алгоритм "Нахождение обратной матрицы"

- Scalar 32: 218 наносекунд.
- SIMD-инструкций: 136 наносекунд
- Ускорение в 1.6 раз



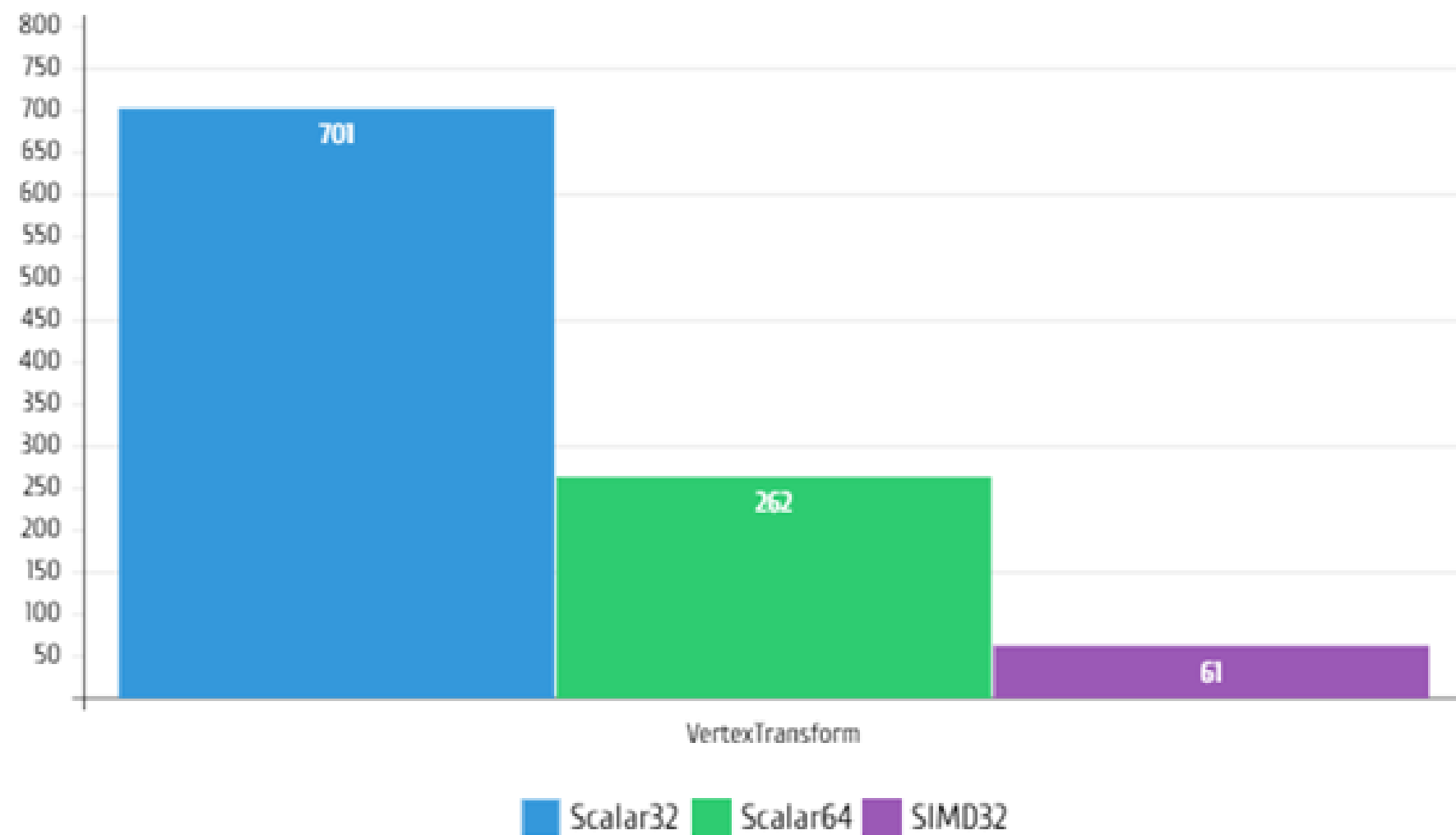
Алгоритм "Умножение матрицы на вектор"

- Scalar 32: 53 наносекунд.
- SIMD-инструкций: 52 наносекунд
- Ускорение незначительное



Алгоритм "Транспонирование матрицы"

- Scalar 32: 20 наносекунд.
- SIMD-инструкций: 14 наносекунд
- Ускорение в 1.4 раз



Алгоритм "Преобразование вершины"

- Scalar 32: 701 наносекунд
- SIMD-инструкций: 61 наносекунд
- Ускорение в 11 раз



ЗАКЛЮЧЕНИЕ

- Применение SIMD позволяет значительно ускорить выполнение алгоритмов и повысить общую производительность кода
- Эффективность применения SIMD может зависеть от конкретных алгоритмов, типов данных и характеристик аппаратной платформы.
- Проведенный анализ и оптимизация подтверждают их значимость и потенциал для улучшения производительности и эффективности вычислений