

Сравнительный анализ RISC-V и Intel64 на базе бенчмарка финансовых вычислений STAC-A2

В. В. Патокова¹, А. Ф. Николаев¹

¹ПАО «Сбербанк»

29 сентября 2026

План доклада

- Постановка задачи
- Реализация
- Оптимизации
- Сравнительный анализ
- Заключение

Постановка задачи

STAC-A2 —бенчмарк на базе реальных задач анализа рыночных рисков, разработанный STAC Benchmark Council

- Широко используется в финансовой индустрии для сравнения программно-аппаратных решений
- Создает большую нагрузку по вычислительным ресурсам и памяти

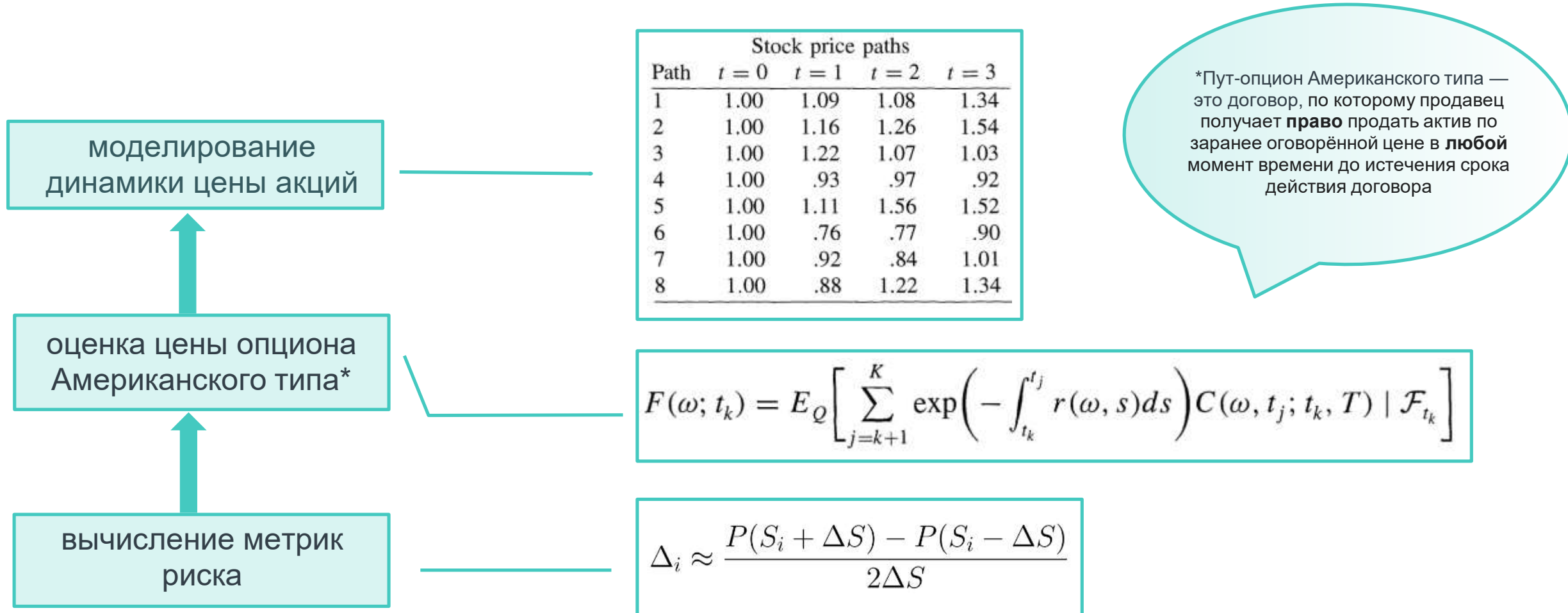
Проблема: Нет реализации для архитектуры RISC-V

Цели работы:

- Разработать референсную реализацию элементов STAC-A2
- Подтвердить функциональную корректность реализации
- Оптимизировать под архитектуру RISC-V:
 - Алгоритмические оптимизации
 - Векторизация кода
 - Параллелизация с использованием OpenMP
- Замерить производительность STAC-A2 на выбранных реализациях RISC-V и Intel 64
- Провести сравнительный анализ выбранных реализаций RISC-V и Intel 64 на основе полученных результатов

Постановка задачи: STAC-A2

Спецификации STAC-A2 ставят **задачу**:



Реализация: моделирование динамики цены

Модель Хестона:

- Моделируется большое количество траекторий, описывающих динамику цен акций и волатильностей
- Выход: массив траекторий цен акций и волатильностей по времени

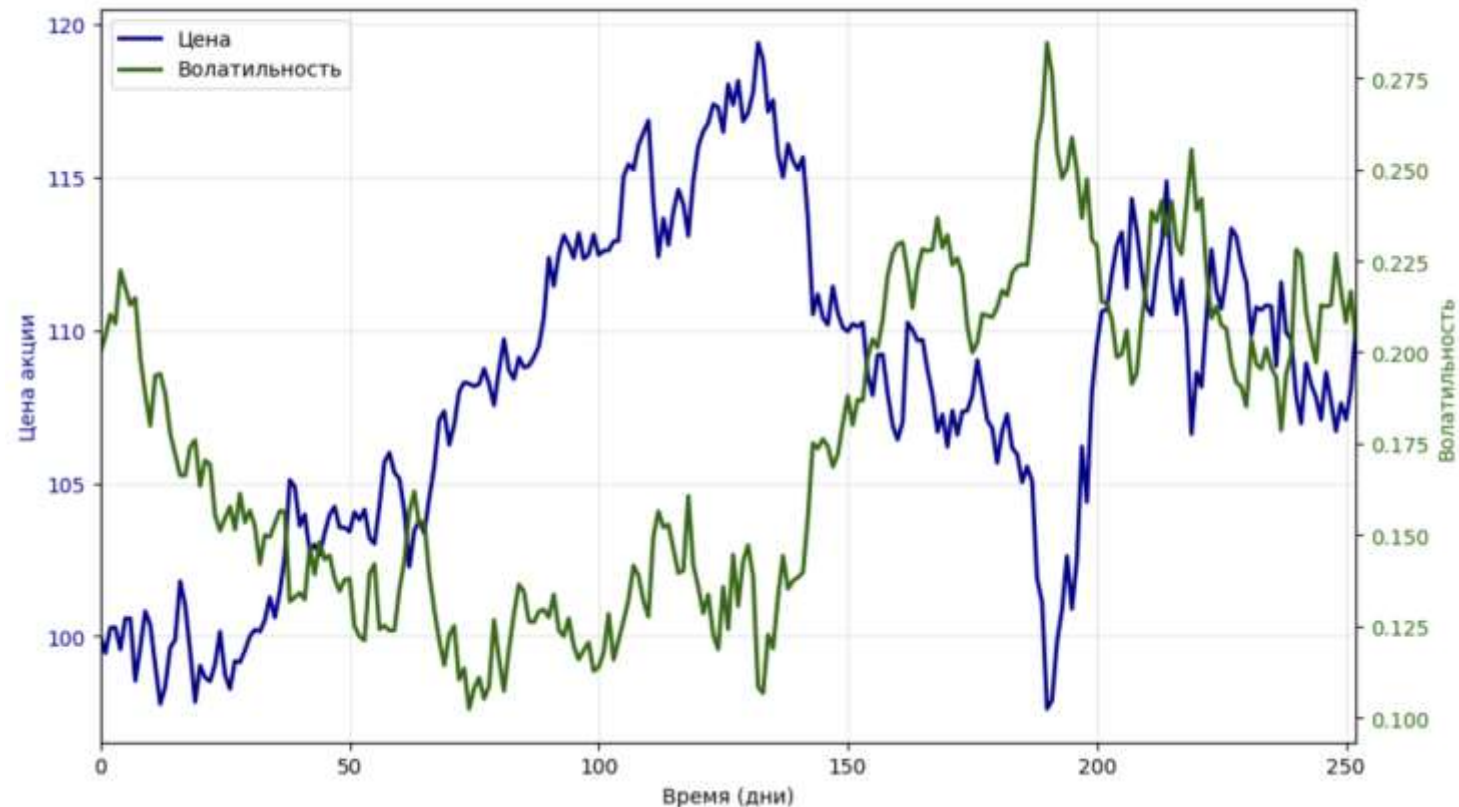
Спецификации требуют:

- IEEE754 fp64
- 5 активов
- 252 шага по времени
- 25000 траекторий

$$\Delta S = \mu S \Delta t + \sigma S \sqrt{\Delta t} \mathcal{N}(0, 1)$$

$$\Delta \sigma^2 = \kappa(\theta - \sigma^2) \Delta t + \xi \sigma \sqrt{\Delta t} \mathcal{N}(0, 1)$$

Симуляция в модели Хестона



Реализация: оценка цены опциона

Спецификации требуют:

- опцион на несколько активов
- используется самый дешевый актив в каждый момент времени $S_t = \min_{i=1..5} S_t^i$

Общая идея алгоритма: Для каждой траектории находим оптимальное время исполнения опциона, вычисляем прибыль, затем усредняем прибыль по всем траекториям

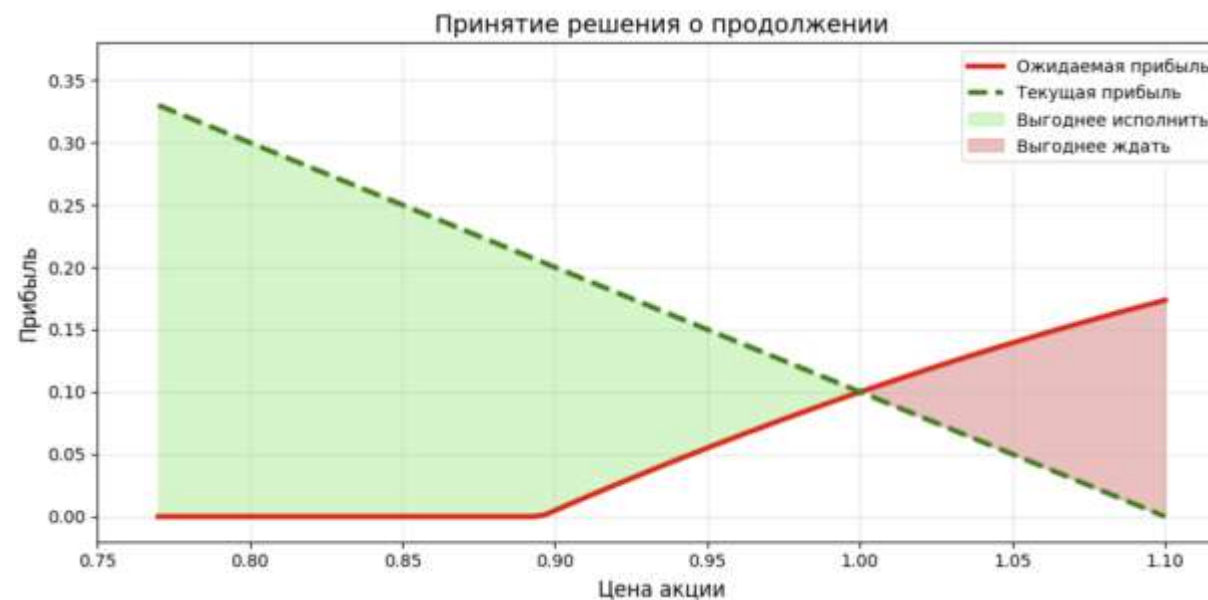
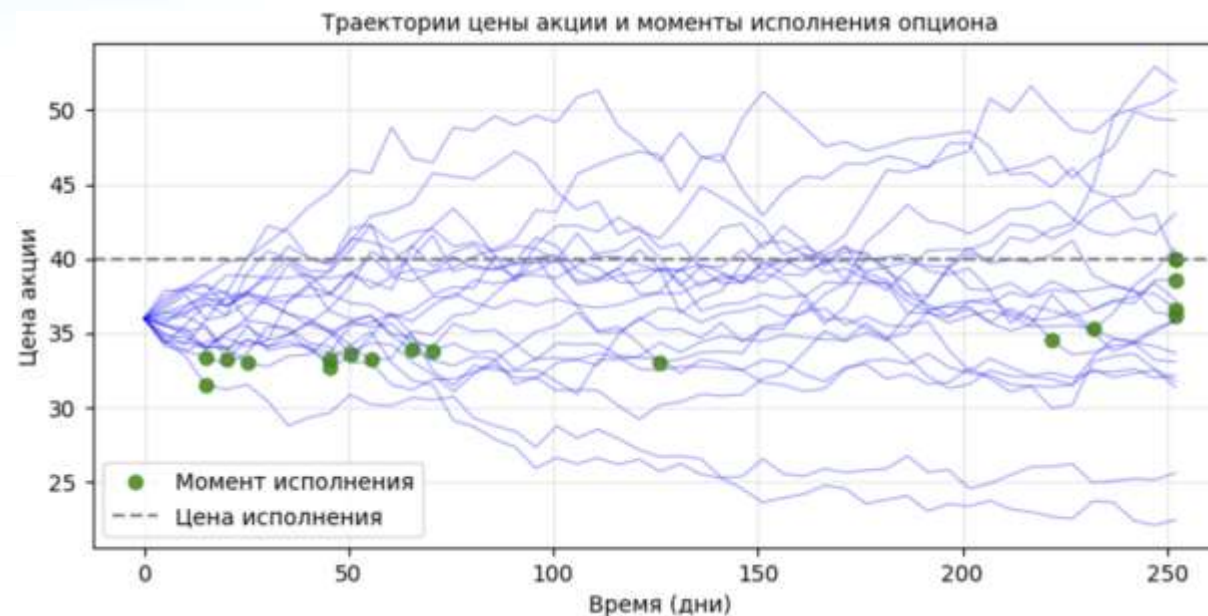
Поиск оптимального времени исполнения: На каждом шаге находим прибыль от исполнения опциона

$$P_{\text{curr}} = \max(K - S_t, 0)$$

и с помощью линейной регрессии оцениваем ожидаемую прибыль от продолжения P_{exp}

$P_{\text{curr}} > P_{\text{exp}}$   Исполняем

$P_{\text{curr}} \leq P_{\text{exp}}$   Ждем



Реализация: вычисление метрик риска

Метрики риска характеризуют чувствительность цены опциона к изменению начальных параметров

Метрика риска	Определение	Формула
дельта	Чувствительность к изменению начальной цены акции	$\frac{\partial P}{\partial S}$
гамма	Чувствительность дельты к изменению цены акции	$\frac{\partial^2 P}{\partial S^2}$
ро	Чувствительность к изменению процентной ставки	$\frac{\partial P}{\partial r}$

Реализация: валидация

Этап валидации	Способ валидации
Моделирование динамики цены акции в модели Хестона	Распределение волатильности: $\Pr(V(T) < x V(t)) = F_{\chi^2} \left(\frac{x \cdot n(t, T)}{e^{-\kappa(T-t)}}; d, V(t) \cdot n(t, T) \right)$
Оценка цены опциона в модели Хестона	Оценка попадания в доверительный интервал, где P_{true} — истинное значение, D — дисперсия, N — число смоделированных траекторий, t — коэффициент Стьюдента: $P_{true} - t \cdot \frac{\sqrt{D}}{\sqrt{N}} < P < P_{true} + t \cdot \frac{\sqrt{D}}{\sqrt{N}}$ ➤ Европейский опцион: сравнение с аналитической формулой Хестона [1] ➤ Американский опцион: сравнение с эталонными значениями из [2]
Вычисление метрик риска	Сравнение с аналитическими значениями в модели Блэка-Шоулса [3]. Оценка попадания в доверительный интервал.

[1] URL: https://www.ma.imperial.ac.uk/~ajacque/IC_Num_Methods/IC_Num_Methods_Docs/Literature/Heston.pdf

[2] DOI: [10.1002/num.20239](https://doi.org/10.1002/num.20239)

[3] URL: <https://www.journals.uchicago.edu/doi/epdf/10.1086/260062>

Оптимизации: алгоритмические оптимизации

Минимизация используемой памяти

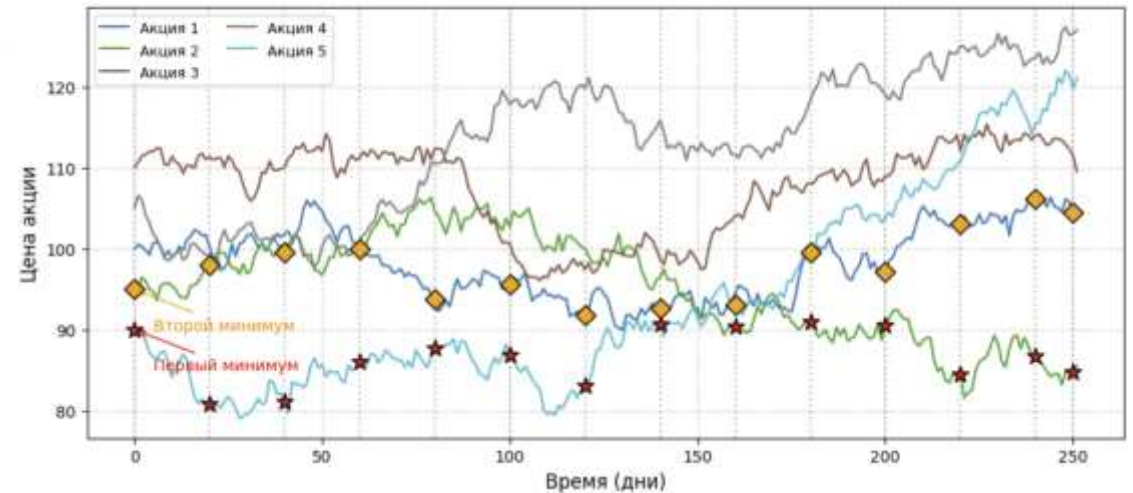
Для оценки цены опциона используется самый выгодный актив — достаточно знать минимальную цену по всем активам

При вычислении метрик риска начальные значения изменяются — минимум также меняется.

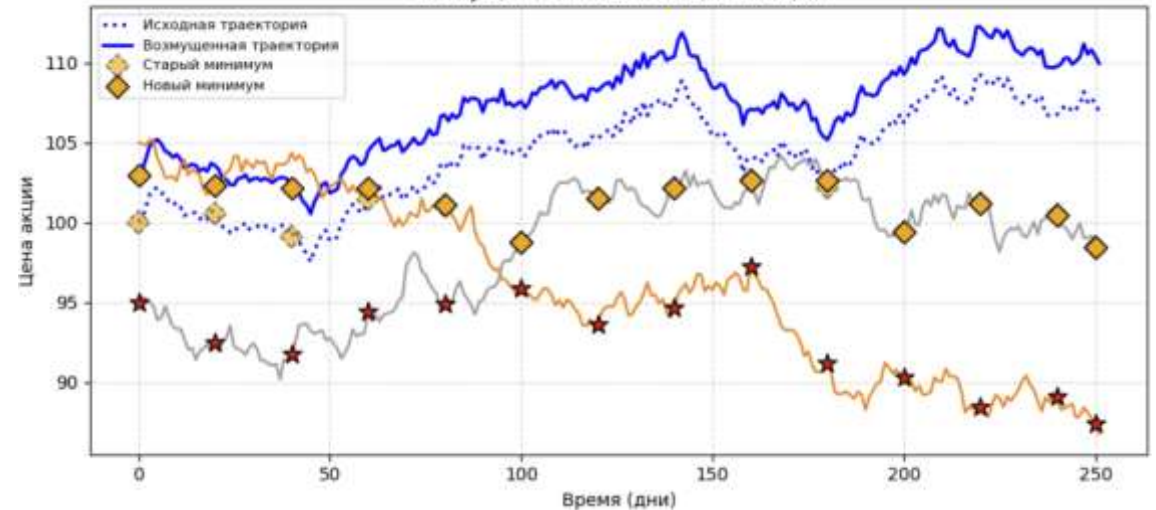
Идея: для нахождения нового минимума достаточно хранить только **две** старые минимальные цены, а не цены все активов

Результат: объем используемой памяти уменьшился в **2,5** раза (с **240** Мб до **96** Мб)

Поиск двух наименьших значений



Возмущение начальной цены акции



Оптимизации: алгоритмические оптимизации

Снижение объема вычислений при расчете метрик риска

При изменении начальной цены акции в каждый момент времени цены на одной траектории связаны соотношением:

$$S_t^{\text{new}} = S_t^{\text{old}} \cdot \frac{S_0^{\text{new}}}{S_0^{\text{old}}}$$



Не требуется заново моделировать траектории

Динамика волатильности не зависит ни от начального значения цены акции S , ни от процентной ставки r :

$$\Delta\sigma^2 = \kappa(\theta - \sigma^2)\Delta t + \xi\sigma\sqrt{\Delta t}\mathcal{N}(0,1)$$



При вычислении метрик риска значения волатильности не меняются, их не надо каждый раз пересчитывать

получено
ускорение в
2,9 раз (с
188 сек до
65,3 сек)

Оптимизации: векторизация серийного кода

Референсная реализация использует стандартные математические функции (libm), генераторы случайных чисел из библиотеки C++ <random> и реализованное вручную матричное умножение

Векторизованы:

- Математические функции (корень, экспонента, логарифм)
- Матричное умножение с использованием библиотеки OpenBLAS
- Генератор нормальных случайных чисел, mt19937
- Моделирование динамики цен акций

Результат: получено ускорение в **2,5** раз

конфигурация: 64 ядра, 2.6 ГГц, L1: 128 Кб, L2: 2 Мб, L3: 64 Мб, RAM 135 Гб, clang 17.0.6, openEuler 24.03, OpenBLAS 0.3.28
опции построения: -O3 -march=rv64gcv_zvl128b -ffast-math



```
compute_greeks.cpp:158:9: remark: vectorized loop (vectorization width: 4, interleaved count: 4) [-Rpass=loop-vectorize]  
for (int j = 0; j < N_PATHS; j++)
```

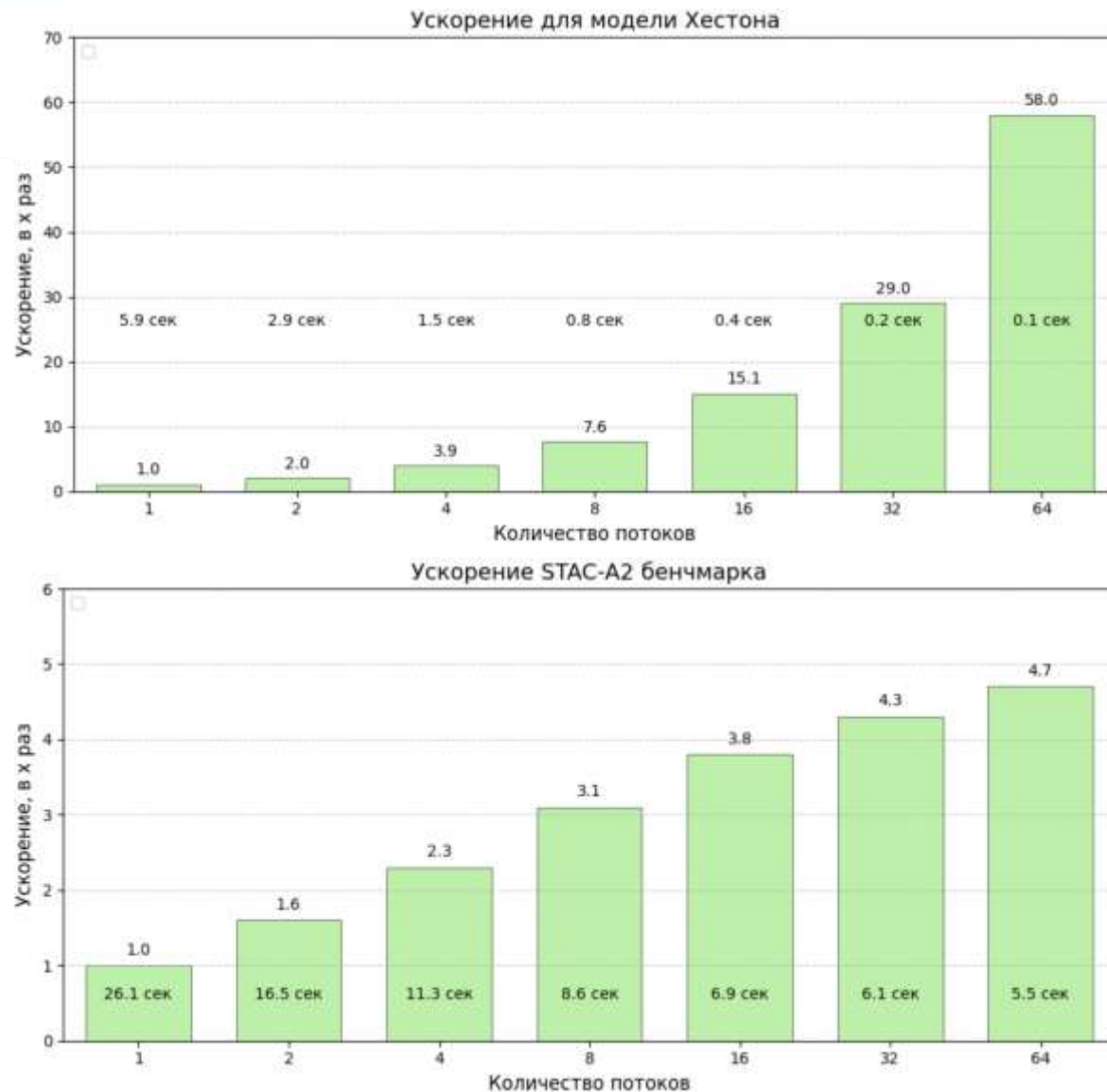
Оптимизации: параллелизация

Параллелизованы:

- Модель Хестона
- Векторные математические функции
- Матричное умножение
- Генератор mt19937: инициализирован для параллельных вычислений с помощью метода skip Ahead

Результат: получено ускорение в **4,7** раз (с **26,1** сек до **5,5** сек)

конфигурация: 64 ядра, 2.6 ГГц, L1: 128 Кб, L2: 2 Мб, L3: 64 Мб, RAM 135 Гб,
clang 17.0.6, openEuler 24.03, OpenMP 5.1
опции построения: -O3 -march=rv64gcv_zvl128b -ffast-math -fopenmp



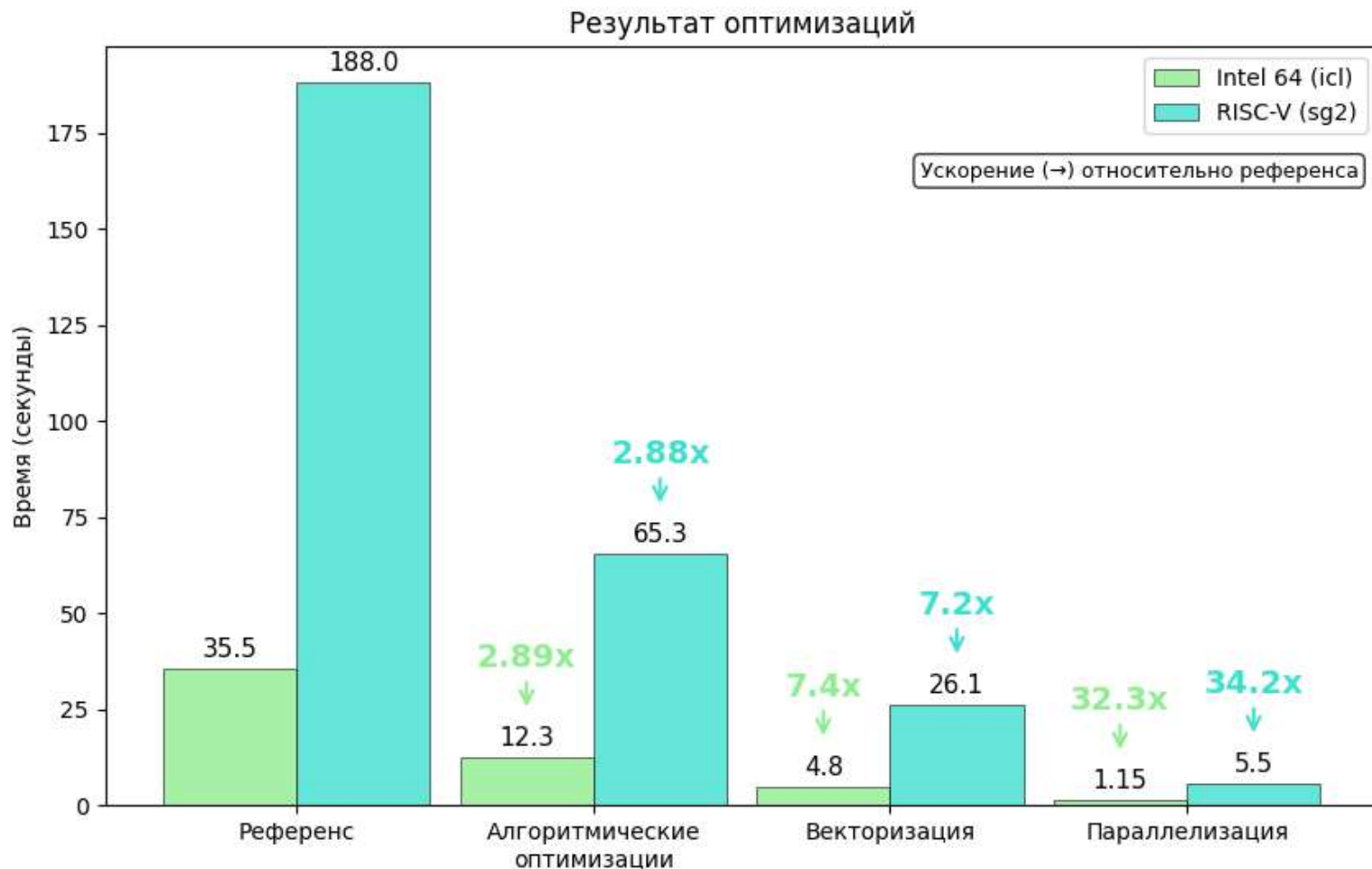
Оптимизации: результаты

Делятся на 3
последовательные стадии:

Алгоритмические оптимизации: позволили сократить объем требуемой памяти в 2,5 раз и увеличить скорость работы в 2,9 раз

Векторизация: обрабатываем до 8 траекторий одновременно за счёт векторных инструкций процессора — расчет цен, генерация случайных чисел, мат.функции и матричное умножение.

Параллелизация: распределяем все траектории между ядрами с помощью OpenMP, каждое ядро работает со своим независимым генератором случайных чисел.



конфигурация sg2 (RISC-V): 64 ядра, 2.6 ГГц, L1: 64 Кб, L2: 2 Мб, L3: 64 Мб, clang 17.0.6, openEuler 24.03, OpenBLAS 0.3.28, OpenMP 5.1

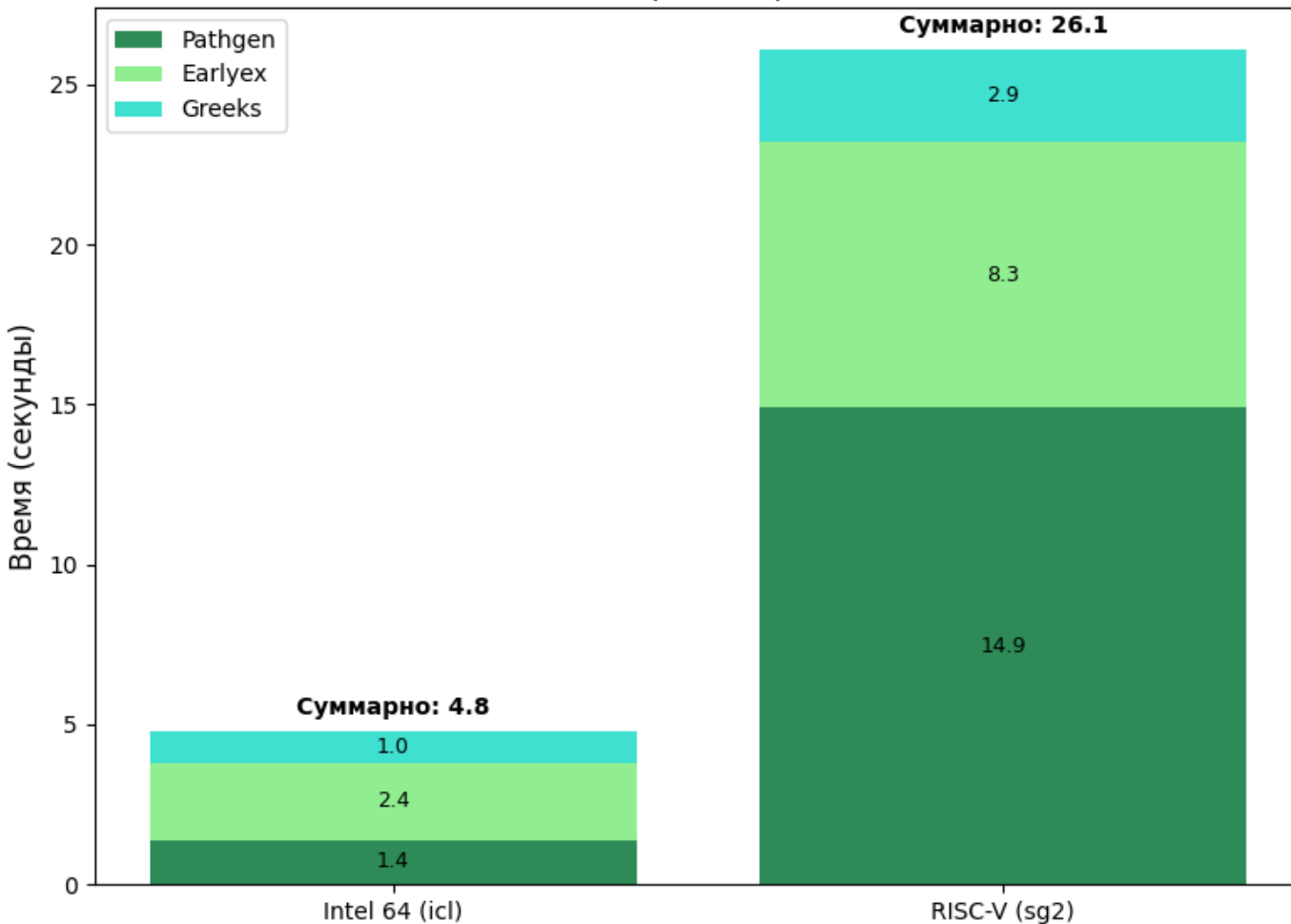
опции построения sg2 (RISC-V): -O3 -march=rv64gcv_zvl128b -ffast-math -fopenmp

конфигурация icl (Intel 64): 32 ядра, 2.9 ГГц, L1: 47 Кб, L2: 1,25 Мб, L3: 24 Мб, clang 16.0.6, Fedora 38, Intel MKL 2025.0, OpenMP 5.0

опции построения icl (Intel 64): -O3 -march=native -ffast-math -fopenmp

Сравнительный анализ для серийной версии

Вклад этапов бенчмарка в производительность



*Описание этапов бенчмарка — на слайдах 5, 6, 7

Пропускная способность векторных инструкций:

Инструкция	icl, GOps/s	sg2, GOps/s
vfmadd	46,4	10,4
vfmul	46,4	10,4
vfsb	46,4	10,4
vfdv	1,45	0,26

средняя
разница **5x**

Пропускная способность подсистем памяти:

	L1, GB/s	L2, GB/s	L3, GB/s	RAM, GB/s
sg2, load	37	16	7	5
sg2, store	19	9	3	2
icl, load/store	198	58	24	13

средняя
разница **3,9x**

Методика измерений

- **Инструкции:** собственные бенчмарки
- **Подсистемы памяти:** STREAM, likwid

Сравнительный анализ для серийной версии

Этап Pathgen **compute-bound**

Типичная часть вычислений (данные в регистрах):

```
vdouble v4_vec = vfmadd(v1_vec, v2_vec, v3_vec);
```

```
vdouble v5_vec = vfmul(v3_vec, v1_vec);
```

```
vdouble v6_vec = vfdiv(v1_vec, v2_vec);
```

Операции: 1 fma, 1 mul, 1 div

Предпосылки:

- На одну загрузку/сохранение данных в кэш в среднем приходится 34 вычислительные инструкции
- Промежуточные вычисления используют только регистры

Вывод: можно пренебречь временем на пересылку данных

$$\text{sg2 compute time} = (2 / 10,4 + 1 / 0,26) * 2,6 = 10,5 \text{ cpe}$$

$$\text{icl compute time} = (2 / 46,4 + 1 / 1,45) * 2,9 = 2,125 \text{ cpe}$$

$$\text{time ratio} = 10,5 / 2,125 = 4,9x$$



Разница в скорости работы **Pathgen** объясняется разницей в скорости работы инструкций

Этапы Earlyex и Greeks **memory-bound**

Типичная часть вычислений (данные в L2 (Earlyex) / RAM (Greeks)):

```
vdouble v1_vec = vload(&v1);
```

```
v1_vec          = vfmul(v1_vec, const);
```

```
v1              = vstore(v1_vec);
```

Операции: 1 load, 1 mul, 1 store

Earlyex:

$$\text{sg2 memory time} = (8 / 16 + 8 / 9) * 2,6 = 3,6 \text{ cpe}$$

$$\text{sg2 compute time} = 1 / 10,4 * 2,6 = 0,25 \text{ cpe}$$

$$\text{icl memory time} = (8 / 58 + 8 / 58) * 2,9 = 0,8 \text{ cpe}$$

$$\text{icl compute time} = 1 / 46,4 * 2,9 = 0,0625 \text{ cpe}$$

$$\text{time ratio} = 3,6 / 0,8 = 4,5x$$

Greeks:

Аналогичные расчеты, но с поправкой на скорость доставки данных из RAM

$$\text{time ratio} = 3,2x$$



Разница в скорости работы **Earlyex** и **Greeks** полностью объясняется разницей в пропускной способности подсистем памяти

*выделено цветом: скорость работы **инструкций**, **загрузки**, **сохранения**, **тактовые частоты** на соответствующих платформах (см. слайд 14)

**cpe (clock per element) — число тактов на элемент векторного регистра

Заключение

Для достижения паритета с icl на sg2 необходимо:

- увеличить пропускную способность инструкций в 5 раз (см. слайд 14)
- увеличить пропускную способность подсистем памяти в 3,9 раз (см. слайд 14)

*Выводы сделаны для реализаций двух архитектур, не для самих архитектур. Для других реализаций результаты будут иными.

В качестве следующего шага предлагается:

провести сравнительный анализ реализаций архитектур для параллельной версии бенчмарка

Разработанная кодовая база и полученные количественные результаты создают методическую основу для дальнейших исследований масштабируемости и применимости реализаций открытой архитектуры RISC-V в критически важных задачах анализа рыночных рисков и других областях высокопроизводительных финансовых вычислений.