УДК 004.415.25

Пятернев Александр Сергеевич, студент
МГТУ им. Баумана

Русакова Зинаида Николаевна, доцент
МГТУ им. Баумана

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ JIT КОМПИЛЯЦИИ ДЛЯ СОКРАЩЕНИЯ ВРЕМЕНИ ФОРМИРОВАНИЯ ПОТОКОВ ПЛАТЕЖЕЙ

Аннотация. В статье рассмотрена технология JIT – компиляции для организации формирования прогнозных потоков платежей для негосударственных пенсионных фондов, обеспечивающая сокращение времени расчетов на основе динамической компиляции программного кода на языке Python.

В данной статье рассмотрена суть и принципы JIT – компиляции, а также ее влияние на производительности приложений и области ее применения. Подробно описано, как JIT – компиляция отличается от других методов оптимизации кода [1].

Далее были описаны основные преимущества и недостатки этой технологии, а также рассмотрены некоторые практические примеры использования JIT-компиляции в различных областях разработки программного обеспечения. Был открыт взгляд на будущее этой технологии и ее потенциальные вклады в дальнейшее совершенствование программирования.

По результатам работы можно сделать вывод, что: JIT – компиляция по скорости выполнения сравнима с компилируемыми языками программирования, JIT компилятор не оказывает влияния на интерпретатор GIL, а также JIT – компиляция позволяет распараллелить выполняемую программу по разным вычислительным кластерам при помощи интерфейса MPI.

Ключевые слова: Формирование прогнозных потоков платежей, JIT компиляция, параллельные вычисления, MPI, Numba, Python, компилируемые и интерпретируемые языки программирования

Введение

В1990-хгодахвРоссиипоявилисьнегосударственныепенсионныефонды, которые стали предлагать свои инвестиционные продукты и перспективу дополнительного дохода к пенсии. Однако, эти фонды столкнулись с проблемой формирования потоков платежей на будущее, так как не было установленной системы взносов и платежей. Кроме того, отсутствие законодательной базы также являлось препятствием, так как не были определены правила взносов в фонды и выплаты пенсий из них.

Начиная с 2000-х годов в России были приняты законы, которые обеспечили формирование системы потоков платежей в негосударственные пенсионные фонды. Федеральный закон” О негосударственных пенсионных фондах принятый в 2002 году, определил правила функционирования этих организаций и установил правила взносов и выплат пенсий из них в соответствии с законодательством.

Почему задача формирования потоков платежей является трудоёмкой? Поскольку у пенсионного фонда может быть около 5 млн. договоров, количество прогнозируемых потоков составляет около 30, а время на которое необходимо сделать прогноз примерно равно 1200 месяцам, что равно 100 лет. Изменяемым здесь является параметр количество договоров и вычислительная сложность задачи в нотации O большое равна O (n) [2].

Новизна данной работы заключается в том, что ранее до появления механизма JIT компиляции никто не воспринимал язык Python для решения высоконагруженных задач, но его с появлением стало возможным не меняя язык программирования получить ускорения выполнения кода.

В рамках данной статьи будут рассмотрены методы сокращения времени формирования потоков платежей на языке программирования Python. Предметом исследования будет механизм JIT компиляции в Python, который позволяет скомпилировать код, написанный на языке Python и запустить его, минуя интерпретатор.

Цель данного исследования: исследовать возможности JIT компиляции в задачи сокращения времени формирования потоков платежей.

Задачи:

– Провести критический анализ JIT компиляции

– Рассмотреть применение JIT компиляции в разных сценариях Разработать методику расчёта прогнозных потоков

– Сделать выводы по полученным данным

Ожидаемый результат данного исследования будет вывод о том, что JIT компиляция в Python позволит значительно сократить время формирования потоков платежей в сравнении с текущей реализацией.

Описание JIT компиляции

Для начала определим, что же такое JIT компиляция? JIT (Just-In-Time) компиляция – это техника исполнения исходного кода, которая позволяет интерпретировать и компилировать программы на лету. Это означает, что исходный код преобразуется в машинный код непосредственно во время выполнения программы. JIT-компиляция может значительно повысить производительность приложений, так как скомпилированный код может быть более эффективно исполнен процессором.

Механизм JIT-компиляции включает следующие этапы:

– Загрузка и анализ исходного кода: При запуске приложения JIT-конвейер анализирует исходный код, чтобы определить его структуру и типы данных. Этот этап также включает в себя проверку на наличие синтаксических ошибок и несоответствий.

– Оптимизация и генерация промежуточного кода (Intermediate Representation, IR): После анализа исходного кода JIT-компилятор выполняет оптимизацию и генерирует промежуточный код. Этот код представляет собой высокоуровневое представление программы, которое уже не содержит синтаксических конструкций конкретного языка программирования. Промежуточный код легко преобразуется в код для различных целевых архитектур.

– Компиляция промежуточного кода в машинный: Промежуточный код компилируется в машинно-зависимый код на данном этапе.

В зависимости от реализации JIT-компилятора, компиляция может выполняться одним из двух способов:

– Транслирующая JIT-компиляция: В этом случае компиляция выполняется каждый раз при обращении к новой функции. Это позволяет быстро начать выполнение программы, но может привести к снижению производительности, если программа содержит большое количество редко используемых функций.

АсинхроннаяJIT-компиляция (Ahead-of-Time, AOT): В этом случае весь код компилируется заранее, и готовые бинарные файлы хранятся в кэше. При первом вызове функции она компилируется, а затем сохраняется в кэше для дальнейшего использования. Это обеспечивает лучшую производительность, но требует больше времени на компиляцию.

– Исполнение скомпилированного кода: Скомпилированный машинный код выполняется процессором. Если во время исполнения возникают ошибки, JIT-компилятор может выполнить дополнительную оптимизацию или откатиться к исходному коду для его дальнейшей интерпретации.

– Завершение работы: После завершения работы программы JIT-конвейер освобождает ресурсы, занятые во время компиляции и выполнения кода.

JIT-компиляция позволяет повысить производительность за счет того, что не требуется предварительная компиляция всего приложения [3]. Правда у этого подхода есть несколько недостатков:

– Замедление запуска приложения: Перед тем, как начать выполнение, JIT-компилятору необходимо проанализировать и оптимизировать исходный код. Это может занять некоторое время, что замедляет запуск приложения.

– Снижение производительности: В некоторых случаях JIT-компиляция может снизить производительность приложения, особенно если оно содержит большое количество редко вызываемых функций. Это происходит потому, что каждый раз, когда вызывается новая функция, JITкомпилятор должен выполнить ее компиляцию, что требует дополнительных ресурсов процессора.

– Проблемы с диагностикой ошибок: Поскольку JIT-компиляция выполняется во время выполнения, ошибки могут возникнуть только после того, как программа уже была запущена. Это затрудняет диагностику и исправление ошибок [4].

– Невозможность оптимизации: JIT-компиляторы обычно не имеют доступа к информации о профиле программы, поэтому они не могут выполнить оптимизацию, которую могли бы выполнить при компиляции перед запуском.

Сложность реализации: Реализация JIT-компиляции может быть сложной и требовать значительных усилий от разработчиков [5].

Результаты проведенных экспериментов

Для того, чтобы оценить эффект от JIT компиляции сравним скорость работы одного и того-же механизма формирования прогнозных потоках, но на разных языках программирования [6]. Для сравнения кода на языке Python с использованием JIT компиляции возьмем язык программирования Си, который по своей сути уже является компилируемым языком программирования. Программа на языке Си скомпилирована с использованием флагов clang -O3 и -march=native для улучшенной оптимизации компилирования кода в машинный язык.

Получаются следующие результаты при выполнении данного кода при разном количестве договоров [7]:

Сложность задачи, тыс. договоров	Скорость выполнения на языке Си, мин	Скорость выполнения на языке Python, мин	Разница относительная	Разница В процентах
1	0,582	0,599	0,017	2,83
5	2,119	2,072	0,047	2,22
10	5,781	5,665	0,116	2,01
50	18,610	18,599	0,011	1,01
100	80,806	80,484	0,322	0,4
500	309,294	309,123	0,171	0,05

Также проведем эксперименты с распараллеливанием при помощи прикладного интерфейса MPI [8]:

Сложность задачи, тыс. договоров	Скорость выполнения на языке Си, мин		Скорость выполнения на языке Python, мин		Разница в процентах
	Последовательный расчёт	Параллельный расчёт	Последовательный расчёт	Параллельный расчёт	
1	0,599	0,690	0,582	0,704	2,75
5	2,072	1,111	2,119	1,157	2,19
50	18,599	4,640	18,610	4,740	1,15
100	80,484	11,989	80,80	12,030	0,7
500	309,123	32,912	309,294	33,076	1,05

Далее приведем графики зависимостей для разных случаев, которые мы рассмотрим дальше [9]:

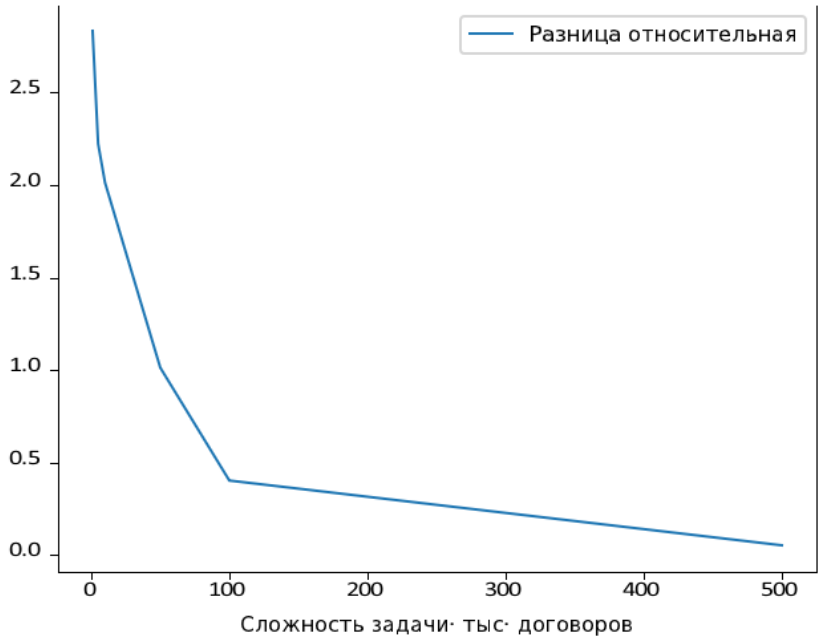

Рисунок 0.1 – График зависимости
скорости выполнения на Python и Си

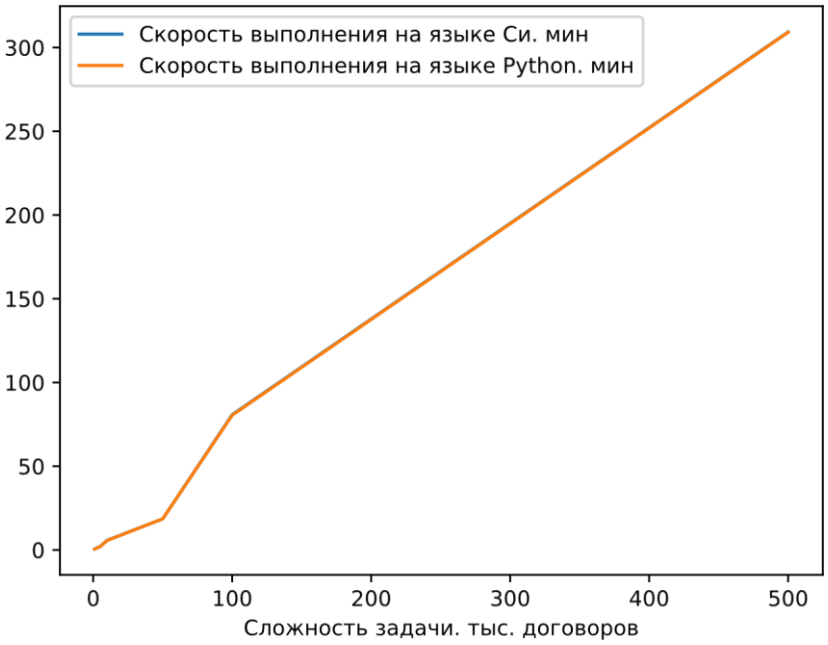

Рисунок 0.2 – Относительная разница
между скоростью выполнения на ЯП Python и Си

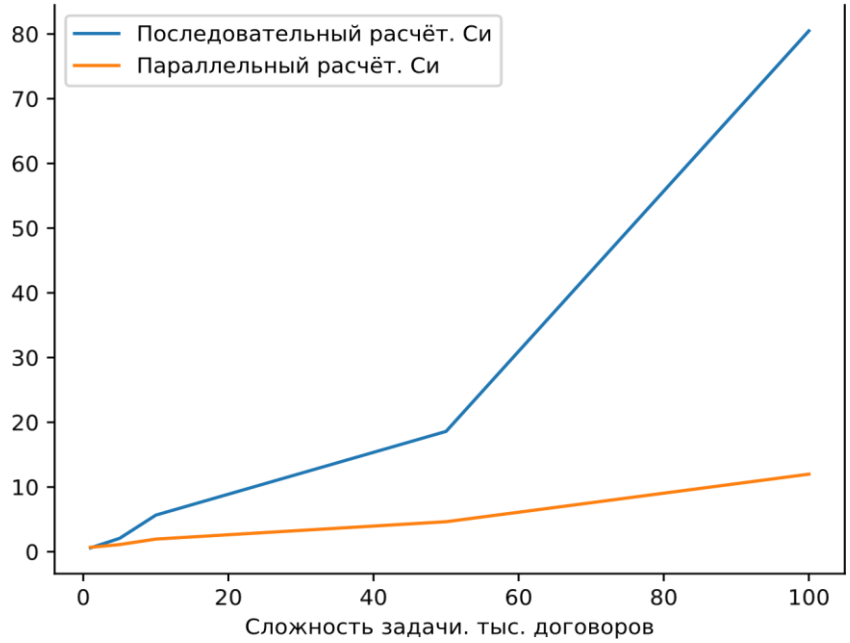

Рисунок 0.3 – Абсолютная разница
между скоростью выполнения на ЯП Python и Си

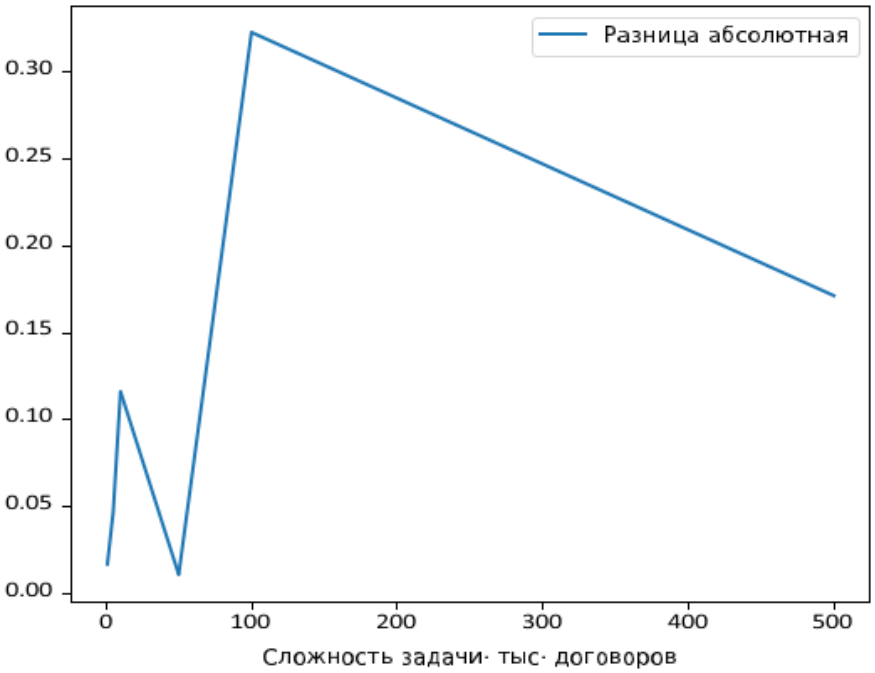

Рисунок 0.4 – Сравнение эффективности
распараллеливания при помощи MPI для ЯП Си

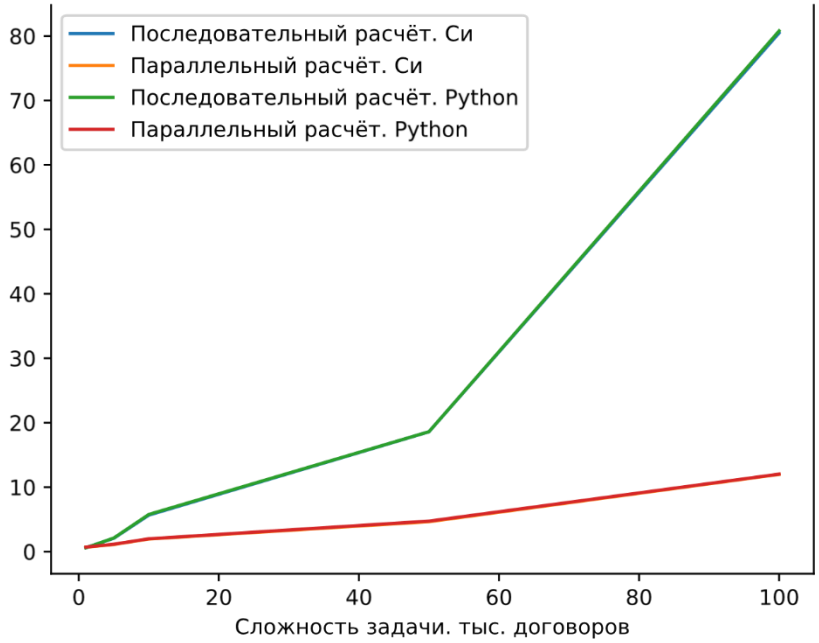

Рисунок 0.5 – Сравнение эффективности распараллеливания при помощи MPI для ЯП Python

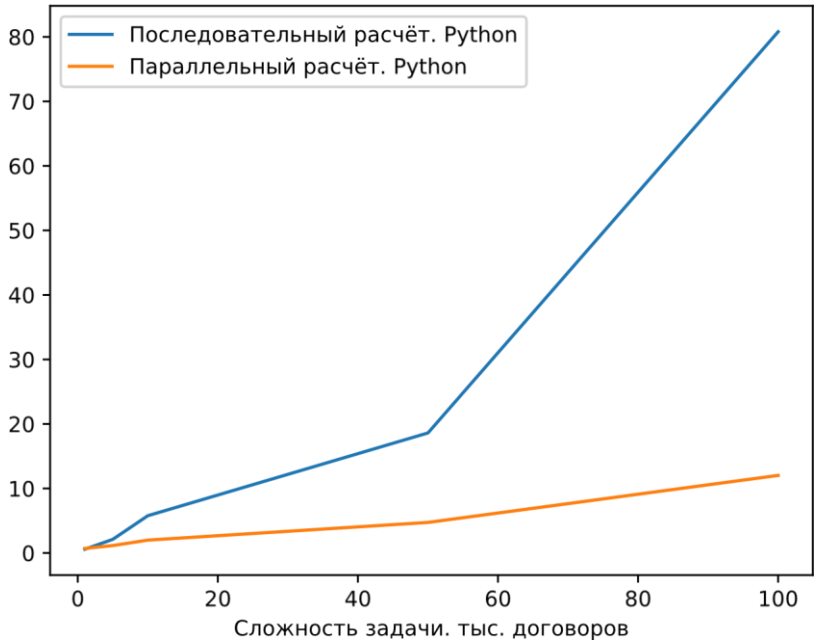

Рисунок 0.6 – Сравнение между собой ЯП Python и Си

Данные для эксперимента были отобраны вычислив среднее значение из 10 проведенных замеров. По результатам всех вычислений получился следующий результат:

– Разница между скоростью получением результата посредством кода, написанным на языке Python с использованием JIT компиляции и кодом, написанном на языке Си составила 3%.

– Порезультатамэкспериментараспараллеливаниедалосокращениеитогового времени близкое к количеству ядер процессора при 500 тыс. договорах. Это свидетельствует о том, что при увеличении количества ядер посредством использования другого процессора или объединения нескольких вычислительных узлов в один кластер коэффициент ускорения вычислений также будет увеличен эквивалентно суммарному количеству ядер всех процессоров при достаточной сложности расчетов.

– Интерпретатор GIL не оказывает существенного влияния на время параллельной работы.

Заключение

В заключение можно отметить, что JIT-компиляция стала неотъемлемой частью современной разработки программного обеспечения, предоставляя эффективный механизм оптимизации кода.

По результатам исследования получилось ускорения кода в 5-10 раз, в сравнении выполнения кода без JIT – компиляции. Также получилась разница в 3% в сравнении с компилируемым языком программирования Си. Такие схожие характеристики делают JIT-компиляцию привлекательным выбором для разработчиков, желающих объединить гибкость интерпретации с эффективностью машинного кода.

Сравнение JIT-компиляции с компилируемым кодом подчеркивает ее важность в контексте современных требований к производительности. Подход с написанием кода на компилируемом языке программирования имеет один существенный недостаток – длительное время написание программного кода.

Подход с написанием кода на интерпретируемом языке программирования лишен этого недостатка, и при этом сохраняется главное достоинство компилируемого кода – высокая скорость выполнения. Но пока существует два недостатка – это проблематичность отладки и невозможность указания специфичной оптимизации инструкций к конкретному процессору.

Несмотря на различия, JIT-компиляция продолжает эволюционировать, предоставляя разработчикам мощный инструмент для достижения оптимальной производительности при минимизации компромиссов в гибкости и удобстве разработки. Следовательно, в будущем можно ожидать дальнейшего расширения областей применения JIT-компиляции и ее интеграции в различные языки программирования и технологические стеки.

Список литературы:

1. Python 3.13 Documentation. – [Электронный ресурс] Режим доступа: https://docs.python.org/3.13/ (дата обращения 07.01.2024).

2. Кочович Е. Финансовый анализ // Москва: Финансы и статистика. – 2014. – С. 223–225.

3. Кокин А. С., Покровский Н. Ю. Факторный анализ дефицитных потоков в негосударственных пенсионных фондах // Аудит и финансовый анализ. – 2023. – С. 30–45.

4. Research GPIF. AI-Driven Liquidity Management for Pension Reserves. – 2022. – [Электронный ресурс] Режим доступа: https://www.gpif.go.jp/ailiquidity-management (дата обращения: 08.06.2025).

5. Committee NAPF Technical. Automated Tools for Fulfilling MCEV Requirements in Pension Funds // NAPF Quarterly Review. – 2022. – С. 22–40.

6. Smith J., Tanaka H. Stochastic Modeling of Pension Liability Cash Flows under IFRS 17 // Journal of Pension Economics and Finance. – 2023. – С. 45–67.

7. Chen L., Dubois M. Monte Carlo Simulations for Long-Term Pension Liability Valuations // Risk Management Magazine. – 2020. – С. 33–49.

8. Eriksson A., Dubois M. Risk-Based Capital Standards and Cash Flow Projections for Japanese Pension Funds. – 2023. – [Электронный ресурс] Режим доступа: https://www.mhlw.go.jp/pension-report-2023 (дата обращения: 08.06.2025).

9. Müller R., Chen L. Implementing IFRS 17 in Non-State Pension Funds: Technical Guide. – 2024. – [Электронный ресурс] Режим доступа: https://www2.deloitte.com/ifrs17-pension-funds (дата обращения: 08.06.2025).