Отчет по Лабораторной работе 10.

Методы оптимизации вычисления кода с помощью потоков, процессов, Cython, отпускания GIL

Выполнила: Чикомазова Алиса (504497)

Цель работы: исследовать методы оптимизации вычисления кода, используя потоки, процессы, Cython и отключение GIL на основе сравнения времени вычисления функции численного интегрирования методом прямоугольников, реализованной на чистом Python.

1. Базовая реализация (Итерация 1)

Для тестов использовалась функция интегрирования методом прямоугольников. В первой итерации код был приведен к стандартам качества: добавлены аннотации типов, документация и тесты (doctest/unittest).

```
..
--------------------------------------
Ran 2 tests in 0.029s

OK
Все doctests пройдены успешно.
Все unittests пройдены успешно.
Iterations      | Avg Time (sec)
10000           | 0.001730
100000          | 0.013658
1000000         | 0.133271
5000000         | 0.737172
0.0031415926535898094
```

Результат: Среднее время выполнения на 10 млн итераций — ~1.3 сек. Код ограничен скоростью интерпретатора и работает строго в одном ядре.

2. Параллелизм: Threads vs Processes (Итерации 2-3)

Был разделен интервал интегрирования на части и распределен между workers.

ThreadPoolExecutor: работает в рамках одного процесса, ограничен GIL. Нет ускорения. Время остается прежним или растет.

ProcessPoolExecutor: создает независимые процессы со своим интерпретатором. Линейное ускорение. Время сокращается пропорционально числу ядер.

```
Запуск тестов для 10000000 итераций.
Method       | Jobs   | Time (sec)
Threads      | 2      | 1.0653
Threads      | 4      | 1.0670
Threads      | 6      | 1.0637
Threads      | 8      | 1.0708
Processes    | 2      | 1.1880
Processes    | 4      | 1.2292
Processes    | 6      | 1.2272
Processes    | 8      | 1.9885
```

Выводы:

Потоки: не дают ускорения (и даже замедляют код) из-за GIL, который не позволяет выполнять байт-код Python одновременно на разных ядрах.

Процессы: масштабируются почти линейно, так как каждый процесс имеет свой экземпляр интерпретатора и GIL.

3. Компиляция и Cython (Итерация 4)

Здесь переписали функцию на Cython, типизировав переменные и заменив вызовы Python-функций на C-аналоги.

Оптимизация C-API:
Использование annotate=True позволило исключить взаимодействие с объектами Python внутри цикла. Это превратило код в чистый C.

Результаты:
Один поток Cython выполняет 10 млн итераций всего за ~0.14 сек.

```
                    Метод  Потоки/Процессы  Время (сек)
Python (Последовательно)                 1     1.758060
Cython (Последовательно)                 1     0.147351
        Cython + Threads                 2     0.150208
        Cython + Threads                 4     0.149674
      Cython + Processes                 2     0.141303
      Cython + Processes                 4     0.165975

Общее ускорение Cython относительно Python: 11.93x
```

4. Снятие GIL и финальная оптимизация (Итерация 5)

Использование блока with nogil в Cython позволило потокам работать параллельно, обходя ограничения стандартного Python.

Сравнение лучших методов (30 млн итераций):

Метод	Workers	Время (сек)
Threads (noGIL)	2	0.346349
Processes	2	0.385946
Threads (noGIL)	4	0.339046
Processes	4	0.402205
Threads (noGIL)	6	0.358944
Processes	6	0.417555

Имеет ли смысл применять мьютексы или семафоры в данной задаче?
В данной реализации — нет, это не имеет смысла и будет вредно для производительности, так как

- архитектура задачи: весь объем вычислений разбит на независимые интервалы, а каждый поток/процесс работает со своими локальными данными и записывает результат в свою локальную переменную acc.
- потоки не пытаются одновременно изменять одну и ту же область памяти в процессе вычислений. Сложение финальных результатов происходит только один раз в главном потоке после завершения всех вычислений.
- если бы мы использовали общий счетчик acc и защищали его мьютексом (lock) на каждой итерации цикла, время выполнения выросло бы. Потоки тратили бы большую часть времени на ожидание очереди и переключение контекста.

Изменятся ли значения времени вычисления в Итерации 2, и нужно ли будет отпускать GIL?
Да, значения изменятся.
Для итерации 2 в версиях Python до 3.12 включительно потоки работали почти так же медленно, как один поток, из-за GIL. В Python 3.14 те же самые потоки смогут исполняться на разных ядрах CPU параллельно.
В Python free-threaded глобального GIL просто не существует. Поэтому специально отпускать его будет не нужно.

Выводы:

Cython + nogil + Threads — самый эффективный метод. Потоки работают так же быстро, как процессы, но потребляют меньше оперативной памяти и быстрее запускаются.

Общие итоги:

Для достижения максимальной производительности необходимо:

1. Следить за временем выполнения, для возможности оптимизации
2. Использовать Cython для критических циклов с полной типизацией.
3. Отпускать GIL (nogil) и использовать ThreadPoolExecutor для масштабирования на многоядерных системах.