Отчет

Лабораторная работа №10. Методы оптимизации вычисления кода

Выполнил: *Голубев Артём Дмитриевич*

ИСУ: *502712*

Группа: *P3124*

Задание

Исследовать методы оптимизации функции численного интегрирования методом левых прямоугольников:

1. Базовая версия на Python (итерация 1).
2. Многопоточная реализация (итерация 2).
3. Многопроцессорная реализация (итерация 3).
4. Оптимизация с Cython (итерация 4).
5. Версия с noGIL и многопоточным запуском (итерация 5).

Сравнить время работы различных итераций сделать выводы

Примечание: отчет составлен до появления четкого технического задания на платформе moodle. Сравнения сделаны по собственному выбору.

Реализация

Итоговая таблица сравнения №1 (Количество прямоугольников (итераций) 10 000 000)

Метод	Время (секунды)	2 потока	4 потока	8 потоков
integrate (последовательно)	0.448273	---	---	---
integrate_async (многопоточность)	---	0.374075	0.378147	0.377016
integrate_async_processes (многопроцессорность)	---	0.251988	0.166861	0.168368
integrate_cy (Cython)	0.372316	---	---	---
integrate_cy_w_cos (Cython со встроенной ф-ей косинуса)	0.023581	---	---	---
integrate_cy_noGIL (многопоточность + noGIL + Cython)	---	0.011931	0.006536	0.006126

Итоговая таблица сравнение №2 (Количество прямоугольников (итераций) 100 000)

Метод	Время (секунды)	2 потока	4 потока	8 потоков
integrate (последовательно)	0.004492	---	---	---
integrate_async (многопоточность)	---	0.003714	0.003791	0.003839
integrate_async_processes (многопроцессорность)	---	0.044203	0.047598	0.067144
integrate_cy (Cython)	0.003668	---	---	---
integrate_cy_w_cos (Cython со встроенной ф-ей косинуса)	0.000255	---	---	---
integrate_cy_noGIL (многопоточность + noGIL + Cython)	---	0.000597	0.000563	0.000617

Для замеров везде была использована тригонометрическая формула косинуса, но для Cython кода были варианты кода, когда функцию можно передать как аргумент, и когда функция фиксирована (использовалась внутренняя функция из языка C). Вычисление было на интервале от 0 до 10 000.

Сравнения

1) Сравним работу чистой функции и оптимизированной с помощью процессов и потоков. Функция с многопоточностью и при малых итерациях, и при больших выигрывает у чистой функции, причем чем больше итераций, тем более заметным становится преимущество. С процесса получается, что при малых итерациях, за счет того, что на распределение и выделение памяти и мощности отдельного процесса тратится не мало времени, многопроцессорный код проигрывает и чистому и многопоточному коду. Но если итераций становится больше, то многопроцессорный подход раскрывает свои возможности и показывает заметно более эффективную работу, чем две другие функции.
2) Сайтонизированный код дает колоссальный прирост как в чистом виде, так и в многопоточном с отключением GIL (noGIL). Если сравнивать их между собой, то ситуация аналогична пункту 1. При малых итерациях многопоточность не дает явной оптимизации, но тратит много времени на подготовку потоков. При больших же итерациях многопоточность выигрывает у чистого сайтонизированного кода, причем более, чем в 2 раза

Вывод

Таким образом, многопоточность и многопроцессорность дают заметный прирост в эффективности лишь при реально тяжелых задачах, так как при малых, код тратит больше время на подготовку процесса/потока. Сайтонизированный код же выигрывает при любых условиях за счет минимизации использования PythonAPI и приближенному использованию С кода (который по определению быстрее кода Python)