Федеральное государственное автономное образовательное учреждение
высшего образования
"Московский физико-технический институт (государственный университет)"
Физтех-школа аэрофизики и космических исследований
Кафедра перспективных технологий для систем безопасности

ОТЧЕТ ПО КУРСУ «ОСНОВЫ ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ»

Студент:
Макаров Николай Романович
Преподаватель:
Акимов Владимир Владимирович,

г. Долгопрудный
2019

Оглавление

Введение

Парадигма параллельного программирования возникла в 50-х годах 20-го века, практически одновременно с появлением первых ЭВМ. Существующие на тот момент компьютеры не могли дать необходимую скорость вычислений, что привело к возникновению идей о разбиении программ на выполняющиеся одновременно части для ускорения вычислений. На данный момент существуют различные способы ускорения вычислений за счет распараллеливания программ: разделение на несколько потоков/процессов, массивно-параллельные вычисления, MPI.

Данный отчет охватывает два метода параллельного программирования: работа программ в несколько потоков и программы с использованием MPI. Многопоточная программа имеет свои особенности, которые обусловлены спецификой языка программирования Python, на котором и выполнены программы. Так как Python является скриптовым языком и изначально рассчитывался только для работы в одном потоке из соображений потокобезопасности, а также ввиду концепта деления одного пространства имён между всеми имеющимися потоками, реализация многопоточных программ в Python выполнена при помощи Global Interpreter Lock (GIL).

GIL отвечает за синхронизацию и очерёдность доступа программы в случае конфликтов имеющихся потоков при попытке получения двумя и более из них доступа к конкретному участку памяти. В данной работе была поставлена задача сравнить идеальную модель распараллеливания программы с реальной ситуацией.

Реализация программы с использованием MPI также имеет свои особенности. В отличии от многопоточных программ, реализацию которых поддерживает непосредственно Python, MPI интерфейс реализован в качестве нескольких сторонних библиотек. В данной работе используется библиотека mpi4py, реализующая все основные методы MPI.

Описание программ

Для многопоточной реализации было решено использовать утилиту из системной библиотеки Python, функцию sleep из модуля time. На вход данная функция принимает число - количество секунд, после чего имплементирует простой процессора заданное время. В процессе разработки такие функции зачастую используются для грубой кодировки синхронизации модулей "вручную например, когда родительскому процессу необходимо подождать инициализации дочернего, или при необходимости отказа в доступе к функционалу на заданное время. Для нас данная функция представляет интерес, так как ввиду её простоты она должна хорошо распараллеливаться и представлять собой идеальный случай

При помощи MPI интерфейса реализовано вычисление нормы вектора заданного типа, по умолчанию - L2. На начальном этапе программы генерируется случайный вектор заданного размера, после чего создается указанное при запуске программы количество нод. Каждой ноде пересылается часть вектора, после чего все ноды обмениваются полученными результатами и каждая из них получает полный квадрат нормы вектора (для этого используется операция allreduce).

Архитектура

Вычисления производились на ноутбуке на процессоре Intel Core i5-7200U 64-bit, размер L1 кеша 128kib, l2 - 512kib, а l3 - 3mib на ОС Ubuntu 16.04.1 LTS. Объём оперативной памяти на ноутбуке составляет 4 Gb, но упереться в её объём наше исследование не должно.

Результаты

Для анализа был произведён замер времени выполнения описанных выше программ при разном количестве потоков/нод MPI с одинаковыми начальными условиями. Результаты представлены на графиках ниже.

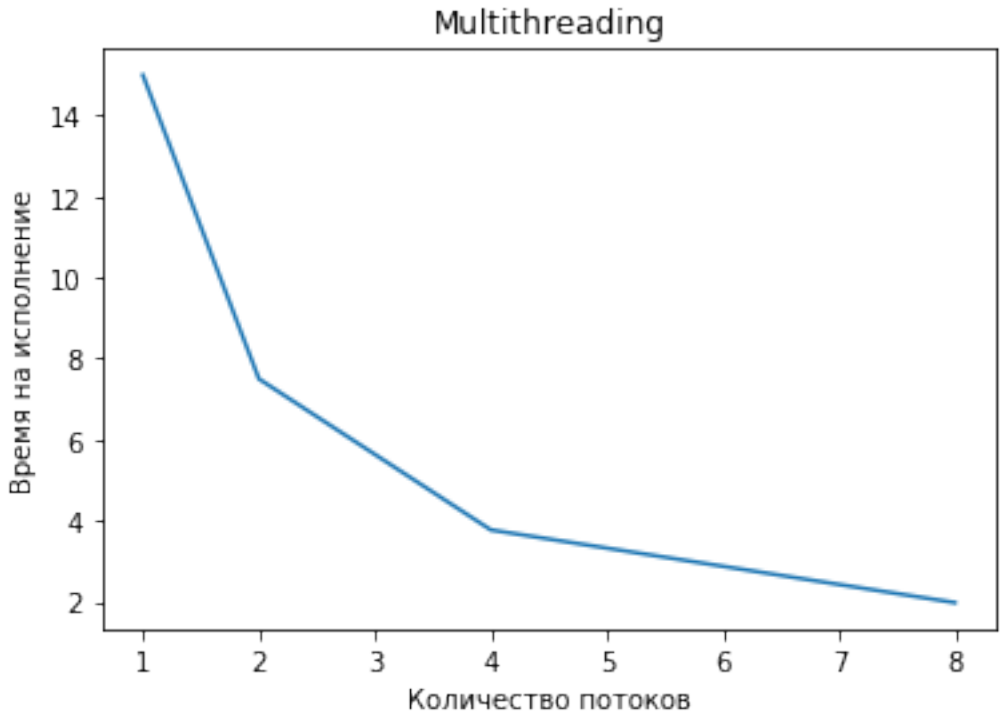

Рис. 1: Время выполнения многопоточной программы от количества потоков

Как и ожидалось, многопоточноя программа показывает линейную зависимость с небольшим отклонением в районе 8 потоков, которое обусловлено растущими затратами процессорного времени на создание и управление потоками.

В случае программы с MPI ассимптотика начинает наблюдаться гораздо более явно и "раньше"относительно числа потоков, нежели чем при исполнении многопоточной программы. Это обусловлено более сложной структурой концепта MPI, а также временными затратами на инициализацию нод и затратами на коммуникацию нод, в многопоточном случае которыми можно пренебречь. Также на данном этапе начинает сказываться относительно небольшой для данного процессора размер L1 - кеша.

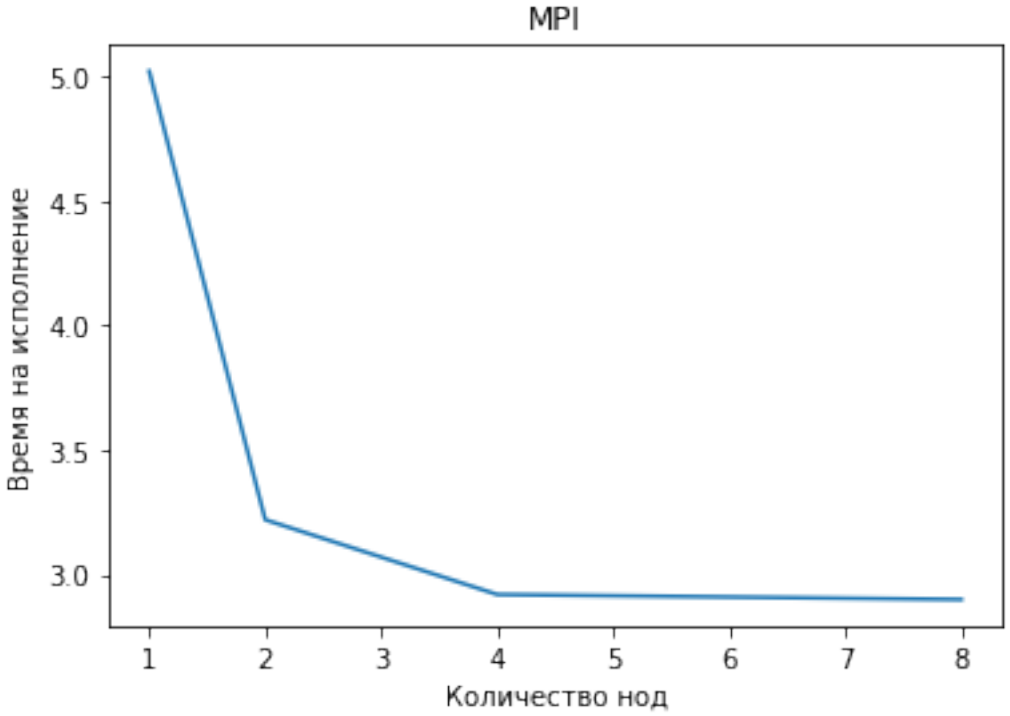

Рис. 2: Время выполнения программы с использованием MPI от количества нод

Заключение

В результате проделанной работы изучены возможности языка Python для параллельного программирования, а также проверен на практике закон Амдаля. Также освоены базовые понятия и операции, используемые для MPI интерфейсов.