Библиотека multiprocessing для распараллеливания и ускорения вычислений

Практикум на ЭВМ 2017/2018

Цыпин Артем Андреевич

МГУ имени М. В. Ломоносова, факультет ВМК, кафедра ММП

21 ноября 2017 г.

Мотивация

- Ускорение вычислений на больших объемах данных

Почему именно multiprocessing?

- Хорошая документация
- Понятный интерфейс
- Основные понятия, использующиеся при работе с библиотекой, знакомы нам из курса «Операционные системы»

Установка библиотеки

В пакете anaconda библиотека уже установлена. Если вы не используете этот пакет, библиотека устанавливается вот так:

```
pip install multiprocess
```

После установки библиотеки, импортируем её:

```
import multiprocessing as mp
```

Постановка задачи

Пусть перед нами стоит задача ускорения подсчета евклидового расстояние между двумя выборками. Расстояние между двумя объектами находится по формуле:

$$\rho(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}$$

Возьмем за основу реализованную в рамках задания [1] функцию `calculate_distance_euclidian(X, Y)`.

Используемые классы и функции

Для распараллеливания упомянутой выше функции нам понадобятся следующие классы и функции:

```
mp.cpu_count()
#количество ядер
mp.RawArray(typecode_or_type, size_or_initializer)
#массив в области общей памяти
mp.JoinableQueue([maxsize])
#очередь с заданиями
mp.Process(group=None, target=None, name=None,
                    args=(), kwargs={}, *, daemon=None)
#объект, который устанавливает, какая функция
#будет исполняться процессом
```

Подробнее об этих классах можно почитать здесь:
https://docs.python.org/3/library/multiprocessing.html

Реализация

Для начала создадим общую область памяти:

```
array = mp.RawArray(ctypes.c_double,
                                    X.shape[0] * Y.shape[0])
```

Реализация

Далее положим в очередь индексы начала и конца подвыборок X, по которым и будет происходить распараллеливание:

```
jobs = [(batch_size * i, batch_size * (i + 1))\
         for i in range(am_jobs)]

queue = mp.JoinableQueue()
    for job in jobs:
        queue.put(job)
    for i in range(mp.cpu_count()):
        queue.put(None)
```

Количество None в конце очереди должно соответствовать количеству работающих параллельно процессов.

Реализация

Создаем процессы. Количество процессов в данном примере равно количеству ядер процессора:

```
workers = []
for i in range(mp.cpu_count()):
    worker = mp.Process(target=func,\
                        args=(X, Y, queue,\
                array, batch_size))
    workers.append(worker)
    worker.start()
    worker.join()
```

Если не использовать метод `mp.Process.join()`, могут возникнуть процессы-зомби.

Реализация

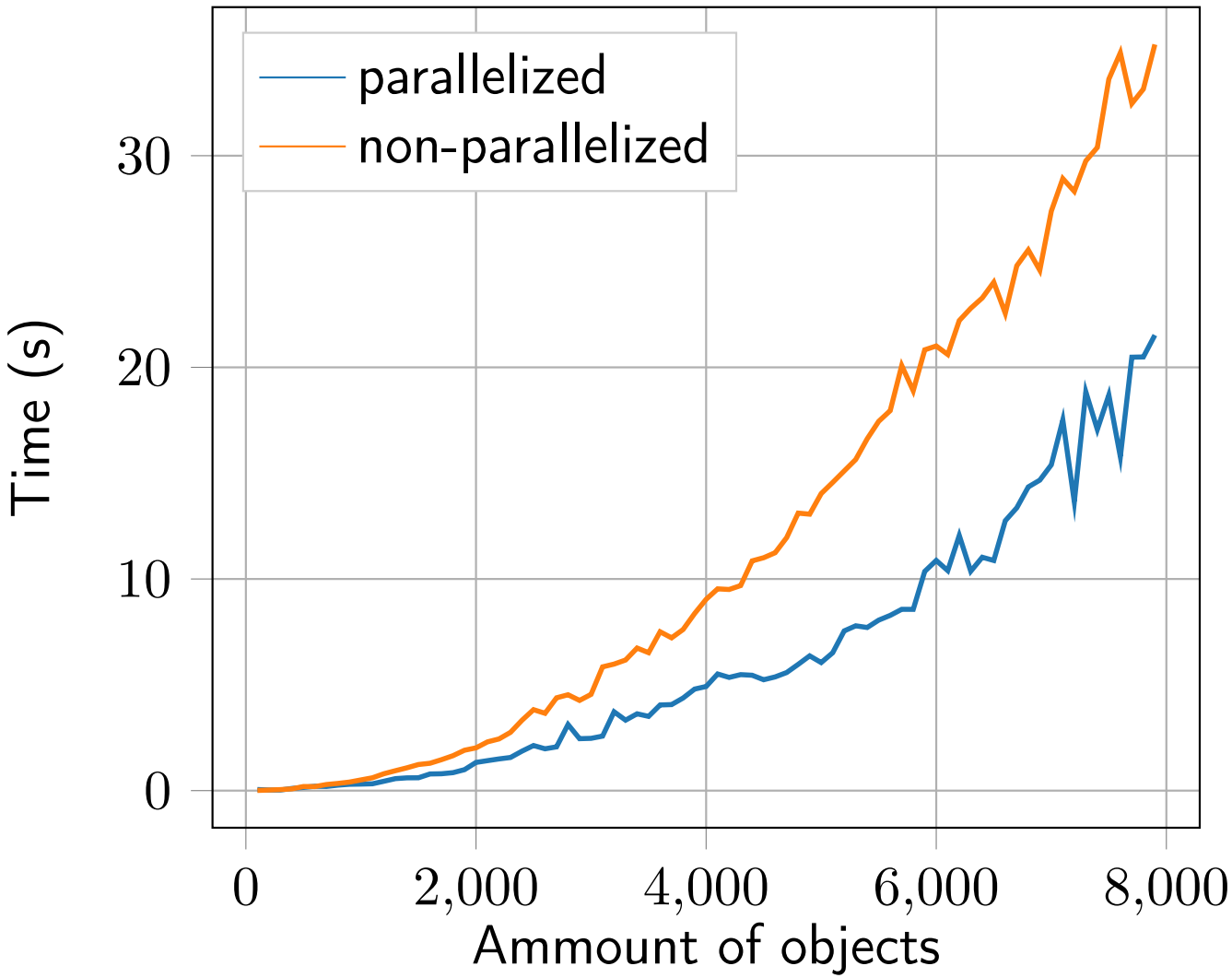
В функции `func()` реализуется подсчет расстояний между подвыборкой X и выборкой Y. Берем задания из очереди, пока не встретим `None` и подсчитываем расстояния:

```
while True:
    job = queue.get()
    if job is None:
        break
    D[job[0]:job[1], :] = \
    calculate_distance_euclidean(X[job[0]:job[1], :], Y)
    queue.task_done()
queue.task_done()
```

Где D - общая область памяти, переведенная в `np.array` для удобства.

Результаты

Working time of parallelized and non-parallelized function



Pool of workers

Напоследок рассмотрим класс `Pool()`:

```
from multiprocessing import Pool
```

Пулл процессов — это очень простой и понятный способ для распараллеливания несложных операций. Например, когда надо в цикле каким-то образом обработать изображения.

Pool of workers

Рассмотрим использование класса Pool() на примере применения Гауссовского фильтра к списку изображений. Нераспараллеленный вариант (код взят из [1]):

```
X_fl = np.array([gaussian_filter(elem.reshape(28, 28),
                 sigma=math.sqrt(1.5)).ravel()
                        for elem in X_train])
```

```
1 loop, best of 3: 3.59 s per loop
```

Pool of workers

Распараллеленный вариант. Переопределим функцию `gaussian_filter()` так, чтобы ее можно было вызывать только от одного аргумента:

```
def func(im, sigma=(1.5 ** (1 / 2))):
        return gaussian_filter(im, sigma)
```

Будем использовать распараллеленный аналог функции `map()` — `Pool.map()`, которая поэлементно применяет функцию к списку.

Pool of workers

Запустим и создадим пул процессов:

```
def gaussian_filter_mp(im):
    pool = Pool()
    res = pool.map(func, im)
    pool.close()
    pool.join()
    return np.array([elem.ravel() for elem in res])
```

Также как и при работе с `mp.Process` необходимо использовать метод `Pool.join()`, чтобы не оставалось процессов-зомби.
Время работы:

```
1 loop, best of 3: 2.72 s per loop
```

Ссылки

- Задание 1. Метрические методы классификации. Практикум 317 группы, https://github.com/arti32lehtonen/mmp_prac_2017/blob/master/Tasks/task1.pdf
- Статья про multiprocessing, https://habrahabr.ru/post/167503/
- Статья про Pool, chriskiehl.com/article/parallelism-in-one-line/