Ускорение вычислений в Python

Использование пакета multiprocessing

Бабичев Дмитрий Олегович

ВМК МГУ

Курс «Практикум на ЭВМ»

Необходимость

При анализе данных очень часто возникает потребность в использовании пользовательских функций, которые, в отличие от большинства билбиотечных функций, не подразумевают автоматического распараллеливания. Пакет multiprocessing позволяет выполнять параллельные вычисления пользовательских функций.

- Виды параллелизма
  - Threads
    - Один вычислительный узел (в нашем случае это ядро)
    - Несколько дочерних процессов
    - Общий ресурс (память)
  - Processes
    - Несколько вычислительных узлов
    - Несколько незывисимых процессов
    - Память нужно разделить между процессами

Задачи

В качестве примера врассмотрим задачи, допускающие независимые вычисления.

- Дана матрица X. Необходима произвести следующие преобразования строк.
  - $f(x) = ||x||^2 + ||x||^3$
  - $f(x) = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})}$
- Дан вектор X. Для каждого x_i из X найти сумму всех простых чисел из $[2, x_i]$

Задачи

Для каждой задачи рассмотрим три решения

- Без параллельных вычислений
- Ручное распараллеливанрие
- Использование "черного ящика"

Задачи

```
1  import multiprocessing as mp
2  import time
3  import numpy as np
4  import ctypes
5  import math
6
7  def f(x):
8      ...
```

Задачи

Решение 1. "В лоб".

```
1   res = []
2   for i in range(X.shape[0]):
3       res.append(f(X[i]))
```

Задачи

Решение 2. Попробуем выполнить вычисления на нескольких ядрах. Для этого будем использовать два метода: MakeWorks и Worker.
MakeWorks принимает на вход объект (матрица или вектор), создает облать общей памяти для результатов, создает очередь заданий и запускает процессы.
Worker принимает на вход объект, список заданий и область общей памяти, производит вычисления и записывает результат. Каждый Worker выполняется отдельным процессом.

Задачи

Метод MakeWorks.

```
1   def MakeWorks(X, nCPU=4):
```

Создадим область общей памяти:

```
1           nRows = X.shape[0]
2       arr = mp.RawArray(ctypes.c_double, nRows)
```

Задачи

Создадим очередь заданий. Каждое задание - диапазон строк в матрице X. None - признак завершения работы процесса.

```
1       nJobs = nCPU * 8
2       q = nRows / nJobs
3       r = nRows % nJobs
4
5       jobs = []
6       firstRow = 0
7       for i in range(nJobs):
8           rowsInJob = int(q)
9           if (r > 0):
10              rowsInJob += 1
11              r -= 1
12          jobs.append((firstRow, rowsInJob))
13          firstRow += rowsInJob
```

Задачи

```
1       queue = mp.JoinableQueue()
2
3       for job in jobs:
4           queue.put(job)
5
6       for i in range(nCPU):
7           queue.put(None)
```

Создадим процессы, запустим их и дождемся выполнения. Возвратим результат.

```
1       processes = [mp.Process(target=Worker, args↩
            =(X, queue, arr))]
2       for p in processes:
3           p.start()
4       for p in processes:
5           p.join()
6
7       return np.frombuffer(arr)
```

Задачи

Метод Worker.

```
1   def Worker(X, queue, arr):
2       results = np.frombuffer(arr)
3       while True:
4           job = queue.get()
5           if job == None:
6               break
7
8           start = job[0]
9           stop = job[0] + job[1]
10
11          """
12          Вычисление для X[start:stop]
13          """
14          results[start:stop] = ... #Результат ↩
                вычислений
```

Задачи

Решение 3. "Черный ящик".

Вместо двух методов из решения 2 можно использовать Pool():

```
1  pool = mp.Pool()
2  res = pool.map(f, X)
3  pool.close()
4  pool.join()
```

map(f, X) применяет функцию f ко всем элементам списка X.

Задачи

Задача 1. $f(x) = ||x||^2 + ||x||^3$
X = np.random.randint(0, 100000, 100000000).reshape(-1, 10)

- Решение 1: 68.721420 second
- Решение 2: 49.657611 second
- Решение 2* :1.469744 second
- Решение 3: 106.332837 second

Задачи

Задача 2. $f(x) = \sqrt{\sum_{i=1}^{n}(x_i - \bar{x})}$
X = np.random.randint(0, 100000, 100000000).reshape(-1, 1000)

- Решение 1: 190.326090 second
- Решение 2: 294.837097 second
- Решение 3: 106.906140 second

Задачи

Задача 3. Для каждого x_i из X найти сумму всех простых чисел из $[2, x_i]$
X = np.arange(100000, 1000000, 100000)

- Решение 1: 47.610485 second
- Решение 2: 47.519706 second
- Решение 3: 31.050579 second

Выводы

Распараллеливание не всегда дает ускорение вычислений. Нужно учитывать:

- Сложность простого вычисления
- Количество задач
- Аргументы

Литература

- Кропотов Д.А. Подготовка презентаций в LaTeXс помощью пакета beamer
  http://www.machinelearning.ru/wiki/images/1/12/MMP_Practicum_317_2015_beamer_presentation.zip
- Пользователь портала Хабрахабр @wagant
  https://habrahabr.ru/post/167503/
- Jesse Noller, Atlanta, 2008
  http://www.slideshare.net/pvergain/multiprocessing-with-python-presentation