УДК 004.853

СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ ПОСЛЕДОВАТЕЛЬНОГО И ПАРАЛЛЕЛЬНОГО АЛГОРИТМОВ НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ PYTHON

Э. Р. Ситников
Донской государственный технический университет (г. Ростов-на-Дону, Российская Федерация)

Рассмотрен язык программирования Python и используемые на нем последовательные и параллельные алгоритмы, определены различия между программами, которые могут быть выполнены параллельно и последовательно, а также скорость компьютера и процент его простоя. Проанализированы результаты совместного использования вышеуказанных алгоритмов.

Ключевые слова: линейный (последовательный) алгоритм, простая передача, параллельное программирование, совместное программирование.

ADAPTATION OF A SEQUENTIAL ALGORITHM TO A PARALLEL ALGORITHM USING PROGRAMMING LANGUAGE PYTHON

Eduard R. Sitnikov
Don State Technical University (Rostov-on-Don, Russian Federation)

The paper considers the Python programming language and the serial and parallel algorithms used on it. The differences between programs are determined that can be executed in parallel and sequentially, their examples, the speed of execution and the percentage of computer idle. We will also consider the joint use of the above algorithms.

Keywords: linear (sequential) algorithm, simple transmission, parallel programming, joint programming.

Введение. В настоящее время перед современными вычислительными системами ставятся задачи, для решения которых требуются большие технические мощности. К примеру, быстрое построение математической модели пятна загрязнения водной среды затруднено из-за большего количества характеристик водоема, которые необходимо учитывать при расчетах. Из-за этого увеличивается время получения результатов, а значит, возникает задержка в реагировании на предотвращение той или иной негативной ситуации. Ускорить данный процесс можно, увеличив производительную мощность вычислительных машин за счет физической модификации. Но модернизация физических составляющих влечет за собой зависимость от производителей компонентов и требует больших экономических затрат. В некоторых случаях незадействованные в расчетах ресурсы могут простаивать. В связи с этим требуется решить актуальную задачу: не произведя физической модернизации, задействовать свободные ресурсы и уменьшить расчетное время операции.

Целью данной статьи является демонстрация процесса повышения производительности вычислительной техники и уменьшения времени, затраченного на расчеты с использованием алгоритмов, выполняющихся параллельно, вместо выполняющихся линейно. Для сравнения алгоритмов использован программный язык Python.

Основная часть. Python — высокоуровневый программный язык общего назначения. Он имеет строгую динамическую типизацию и производит автоматическое управление памятью, что делает его ориентированным на увеличение производительности разработчиков, повышение качества и читаемости кода. Также данный язык улучшает обеспечение переносимости программ, написанных на нём.

Существует два основных алгоритма на Python: линейный и параллельный [1].

Последовательный, или линейный, алгоритм — это нахождение указанного значения произвольной функции на некотором отрезке [2]. Данный алгоритм — это простейшим алгоритм поиска. В отличие, например, от параллельного поиска, он не определяет никаких ограничений на функцию и имеет простую реализацию. Нахождение значения функции осуществляется простым сравниванием значения, которое рассматривается поочередно, и если значения совпадут (с определенной точностью), то поиск завершён [3].

Если скрипты на Python работают медленнее, чем требуется для поставленной задачи, их можно ускорить, запуская параллельные задачи.

Параллельные алгоритмы позволяют выполнить много вычислений одновременно. Это сокращает время выполнения программы. Существует теория, что объём данных, который требуется обрабатывать программами для вычисления, удваивается каждые два года. International Data Corporation дает оценку, что к 2023 году в мире будет обрабатываться 5800 ГБ данных на человека. Такой большой объём данных предъявляет высокие требования к вычислительным мощностям. И параллельные алгоритмы в программировании продолжают оставаться наиважнейшим способом тщательной обработки данных [4].

Один из самых простых способов понимания параллельно алгоритмического программирования — сопоставление его с линейным программированием. В то время как какая-либо последовательная программа постоянно находится в определенном отрезке времени в одном месте, при параллельном написании кода разные его фрагменты являются независимыми либо частично независимыми. Это значит, что в одно и то же время выполнение одного из фрагментов не зависит от результата другого [5]. На рис. 1 показаны основные отличия между линейным и параллельным алгоритмами.

Рис. 1. Отличие параллельного и линейного алгоритмов

Ускорение времени использования — это наиважнейшее преимущество параллельного алгоритма. Часть задач является независимой, поэтому они могут завершиться в одно и то же время. Это позволяет за более короткий промежуток времени выполнить идентичную программу [6].

Допустим, что есть определенная несложная функция, которая проверит какое-либо число (положительное или ноль) простым способом (рис. 2).

```
# Chapter01/example1.py
from math import sqrt
def is_prime(x):
if x < 2:
 return False
if x == 2:
 return True
if x % 2 == 0:
 return False
limit = int(sqrt(x)) + 1
for i in range(3, limit, 2):
 if x % i == 0:
  return False
 return True
```

Рис. 2. Код линейного алгоритма программы

Помимо этого, есть определенный список очень больших целых (от 10^{13} до 10^{13} + 500), и стоит задача выяснить, используя вышеуказанный метод, будет ли каждое из них не сложным:

input = [i for I in range(10 ** 13, 10 ** 13 + 500)].

Какой-либо линейный способ будет простым путем передачи определенного числа за следующим числом в данном методе is_prime() указанным способом (рис. 3):

```
# Chapter01/example1.py
from timeit import default_timer as timer
# sequential
start = timer()
result = []
for i in input:
 if is_prime(i):
  result.append(i)
print('Result 1:', result)
print('Took: %.2f seconds.' % (timer() - start))
```

Рис. 3. Код линейного алгоритма программы передачи простого числа

После того как запускается код, получим вывод (рис. 4):

```
python example1.py
Result 1: [10000000000037, 10000000000051, 10000000000099, 10000000000129,
10000000000183, 10000000000259, 10000000000267, 10000000000273,
10000000000279, 10000000000283, 10000000000313, 10000000000343,
10000000000391, 10000000000411, 10000000000433, 10000000000453]
Took: 3.41 seconds.
```

Рис. 4. Результат выполнения линейного алгоритма

Можно определить, что данный скрипт потребовал примерно 3,41 секунды для вычисления вышеуказанных отобранных данных. Для подтверждения результата запускаем данный сценарий Python при открытом приложении «Монитор активности» в своей операционной системе. На рис. 5 отображены результаты.

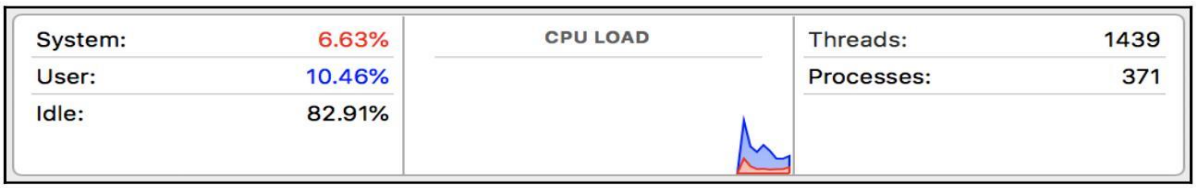

Рис. 5. Вычислительные характеристики

Мы видим, что имеется около 83 % простоя, следовательно, компьютер в данный момент работает, используя не все доступные ресурсы.

Теперь рассмотрим, как параллельный алгоритм повысит производительность нашего компьютера при использовании программы. Данный метод is_prime() включает в себя множество сложных математических действий, следовательно, его можно вычислить с помощью параллельных алгоритмов. Если какая-либо передача определенного значения в функцию is_prime() не зависит от передачи другого числа, то возможно использование параллельности в программе нижеуказанным способом (рис. 6).

```
# Chapter01/example1.py
# concurrent
start = timer()
result = []
with concurrent.futures.ProcessPoolExecutor(max_workers=20) as executor:
 futures = [executor.submit(is_prime, i) for i in input]
 for i, future in enumerate(concurrent.futures.as_completed(futures)):
  if future.result():
   result.append(input[i])
print('Result 2:', result)
print('Took: %.2f seconds.' % (timer() - start))
```

Рис. 6. Код параллельного алгоритма программы

Применяя данный метод, мы получили время выполнения намного меньше (2,33 сек.), причём вычислительная техника использовала большее количество своих компонентов (рис. 7). Простой составил 37 %.

```
python example1.py
Result 2: [10000000000183, 10000000000037, 10000000000129, 10000000000273,
10000000000259, 10000000000343, 10000000000051, 10000000000267,
10000000000279, 10000000000099, 10000000000283, 10000000000313,
10000000000391, 10000000000433, 10000000000411, 10000000000453]
Took: 2.33 seconds
```

Рис. 7. Результат выполнения параллельного алгоритма

Данный результат представлен на рис. 8.

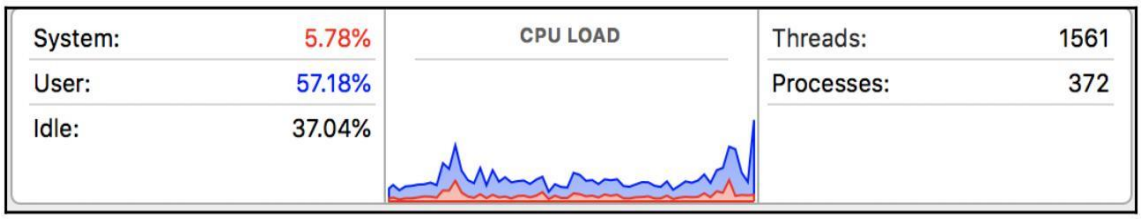

Рис. 8. Монитор активности показывает производительность компьютера

Многие программы могут быть построены на совместно исполняемых алгоритмах. Совместность — это не то же самое что параллелизм. Основным отличием совместимости от параллелизма в программировании является то, что параллельная программа использует какое-либо количество вычислительных процессов, которые работают самостоятельно по отдельности. При этом могут выполняться разные вычислительные процессы, которые используют какой-либо единовременно делимый ресурс в программах, используемых совместно [2].

Так как данный общий источник можно прочитать и переписать любым процессом, потребуется какой-либо метод воздействия в тот временной момент, когда задачи, требующие исполнения, не полностью самостоятельны друг от друга. Иначе говоря, некоторым задачам необходимо выполняться после определенных для того, чтобы программа рассчитала правильный результат.

Заключение. Автору удалось продемонстрировать, что алгоритмы, выполняющиеся линейно, более требовательны к физической возможности производить расчеты, в то время как параллельным алгоритмам необходим больший ресурс. Переход на параллельно выполняющиеся алгоритмы позволяет использовать имеющиеся технические модификации с большей производительностью, что дает возможность уменьшить скорость расчетов. Следовательно, если строение алгоритма предусматривает его программную реализацию в параллельном исполнении,

то это дает возможность получать результаты за более короткий промежуток времени, при этом не потребуется проводить манипуляции с физической составляющей устройств.

Библиографический список

1. Гэддис, Т. Начинаем программировать на Python / Т. Гэддис ; [пер. с англ.]. — 4-е изд. — Санкт-Петербург : БХВ-Петербург, 2019. — 768 с.

2. Зыков, С. В. Программирование. Объектно-ориентированный подход : учебник и практикум для академического бакалавриата / С. В. Зыков. — Москва : Юрайт, 2019. — 155 с.

3. Рейтц, К. Автостопом по Python / К. Рейтц, Т. Шлюссер. — Санкт-Петербург : Питер, 2017. — 336 с.

4. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учеб. пособие для прикладного бакалавриата / Д. Ю. Федоров. — 2-е изд., перераб. и доп. — Москва : Юрайт, 2019. — 161 с.

5. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учеб. пособие для СПО / Д. Ю. Федоров. — Москва : Юрайт, 2019. — 126 с.

6. Шелудько, В. М. Основы программирования на языке высокого уровня Python : учебное пособие / В. М. Шелудько. — Ростов-на-Дону ; Таганрог : Издательство Южного федерального университета, 2017. — 146 с.

Об авторе:

Ситников Эдуард Романович, аспирант кафедры «Математика и информатика» Донского государственного технического университета (344003, РФ, г. Ростов-на-Дону, пл. Гагарина, 1), eduard_sitnikov_96@mail.ru

About the Author:

Sitnikov, Eduard R., Postgraduate student, Mathematics and Computer Science Department, Don State Technical University (1, Gagarin sq., Rostov-on-Don, 344003, RF), eduard_sitnikov_96@mail.ru