Алгоритмы параллельных вычислений. Сортировка слиянием (Algorithm M)

В. С. Верхотуров

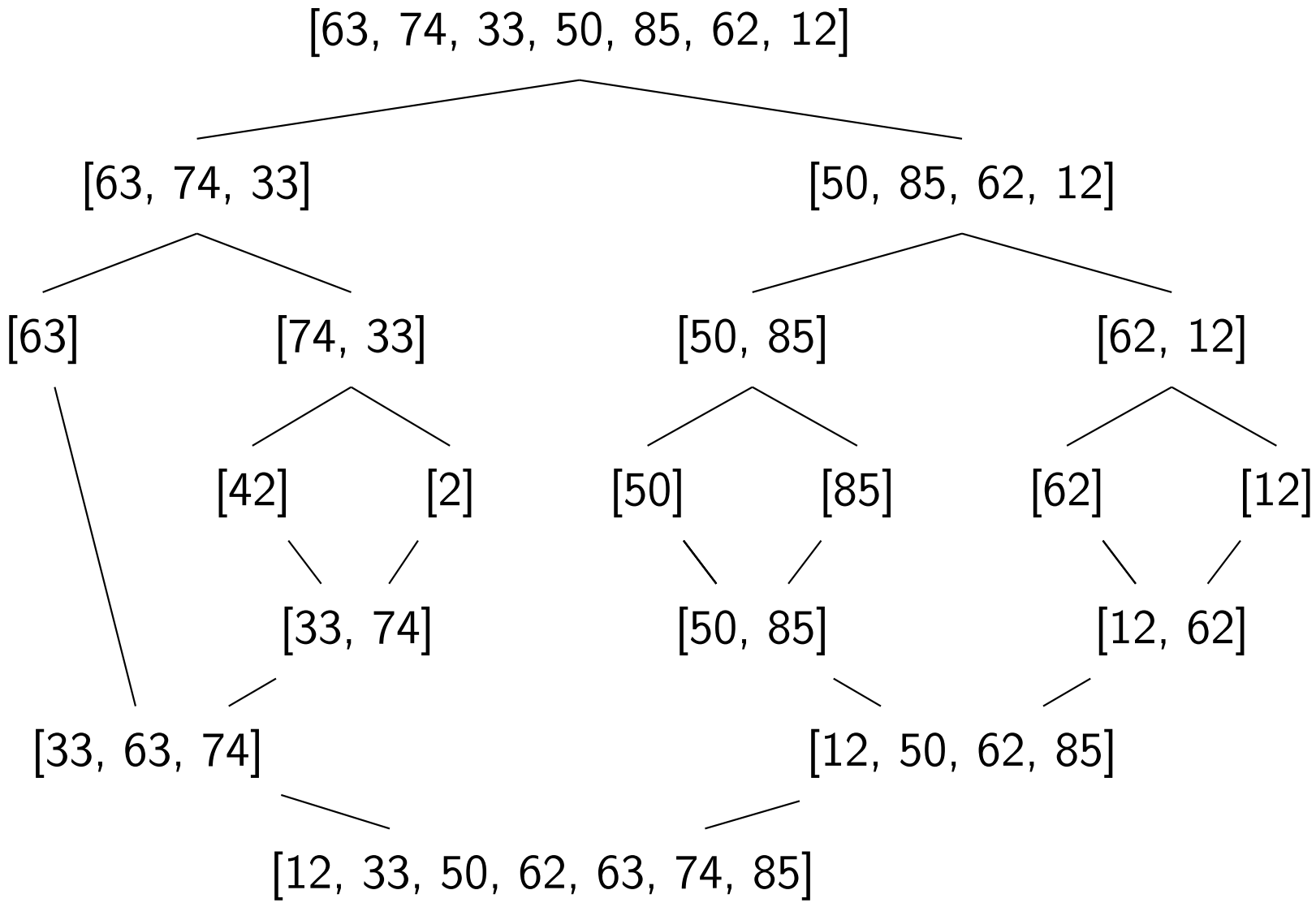
БСБО-05-20
РТУ МИРЭА

7 октября 2022 г.

Содержание

Определение

Сортировка слиянием — это алгоритм «разделяй и властвуй».



$$\begin{cases} 33 & 63 & 74 \\ 12 & 50 & 62 & 85 \end{cases} \tag{1}$$

$$12 \quad \begin{cases} 33 & 63 & 74 \\ 50 & 62 & 85 \end{cases} \tag{2}$$

$$12 \quad 33 \quad \begin{cases} 63 & 74 \\ 50 & 62 & 85 \end{cases} \tag{3}$$

$$12 \quad 33 \quad 50 \quad \begin{cases} 63 & 74 \\ 62 & 85 \end{cases} \tag{4}$$

$$12 \quad 33 \quad 50 \quad 62 \quad \begin{cases} 63 & 74 \\ 85 \end{cases} \tag{5}$$

$$\begin{array}{cccccccc} & & 12 & 33 & 50 & 62 & 63 & \begin{cases} 74 \\ 85 \end{cases} \end{array} \tag{6}$$

$$\begin{array}{cccccccc} & 12 & 33 & 50 & 62 & 63 & 74 & \begin{cases} \\ 85 \end{cases} \end{array} \tag{7}$$

$$\begin{array}{cccccccc} 12 & 33 & 50 & 62 & 63 & 74 & 85 & \begin{cases} \\ \\ \end{cases} \end{array} \tag{8}$$

Реализация однопоточного алгоритма на CPython

```
1  from random import randint
2  from numbers import Number
3
4  def merge(arrays: list[list[Number]]) -> list[Number]:
5    assert len(arrays) == 2
6    x, y = arrays
7    index_x = index_y = 0
8    out = []
9    while index_x < len(x) and index_y < len(y):
10     if x[index_x] < y[index_y]:
11       out.append(x[index_x])
12       index_x += 1
13     else:
14       out.append(y[index_y])
15       index_y += 1
16   out += x[index_x:] + y[index_y:]
17   return out
18
19 def merge_sort(arr : list[Number]) -> list[Number]:
20   if len(arr) <= 1:
21     return arr
22   if len(arr) == 2:
23     return arr if arr[0] < arr[1] else [arr[1], arr[0]]
24   mid = len(arr) // 2
25   return merge(merge_sort(arr[:mid]), merge_sort(arr[mid:]))
26
27
28 def test_merge_sort():
29   input_array = [randint(1, 100) for i in range(10)]
30   print(merge_sort(input_array))
31
32 test_merge_sort()
33
```

Реализация однопоточного алгоритма на CPython. Часть 1

```
1 def test_merge_sort():
2   input_array = [randint(1, 100)
3       for i in range(10)]
4   print(merge_sort(input_array))
5
6 test_merge_sort()
7
```

Реализация однопоточного алгоритма на CPython. Часть 2

```
1  def merge_sort(arr : list[Number]) -> list[Number]: #
       T^{merge}(n)
2    if len(arr) <= 1:
3      return arr
4    if len(arr) == 2:
5      return arr
6        if arr[0] < arr[1]
7        else [arr[1], arr[0]]
8    mid = len(arr) // 2
9    return merge(merge_sort(arr[:mid]),
10             merge_sort(arr[mid:])) # 2 * T^{sort}(n / 2)
11
```

Реализация однопоточного алгоритма на CPython. Часть 3

```
1  def merge(arrays: list[list[Number]]) \
2          -> list[Number]:
3    assert len(arrays) == 2
4    x, y = arrays
5    index_x = index_y = 0
6    out = []
7    while index_x < len(x) and \
8        index_y < len(y):
9      if x[index_x] < y[index_y]:
10       out.append(x[index_x])
11       index_x += 1
12     else:
13       out.append(y[index_y])
14       index_y += 1
15   out += x[index_x:] + y[index_y:]
16   return out
17
```

Вычислительная сложность

Худшая выч. сложность:

$$O(n \log n) \tag{9}$$

Лучшая выч. сложность:

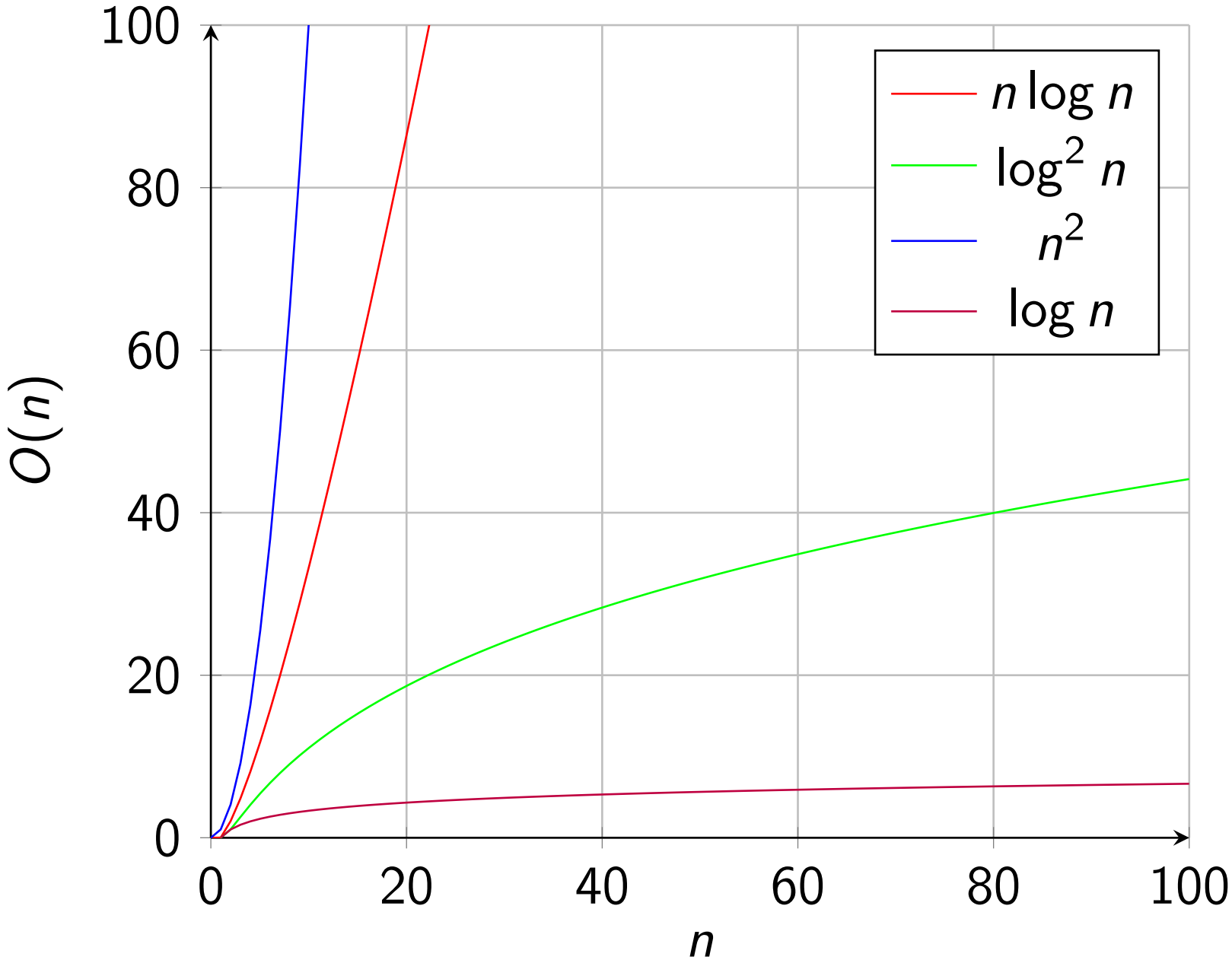
$$O(n \log n) \tag{10}$$

Средняя выч. сложность:

$$O(n \log n) \tag{11}$$

Временная сложность:

$$T^{sort}(n) = 2T^{sort}\left(\frac{n}{2}\right) + T^{merge}(n) = \\ = 2T^{sort}\left(\frac{n}{2}\right) + \Theta(n \log n) \tag{12}$$


100
80
60
40
20
0
O(n)
0
20
40
60
80
100
n
n log n
log^2 n
n^2
log n

Параллельные вычисления в CPython (side-stepping the GIL (Global Interpreter Lock))

Рис.: Иллюстрация GIL

- `multiprocessing.Process`, `multiprocessing.Pool`
  `https://docs.python.org/3/library/multiprocessing.html`,
- C-расширение, расширение на Cython
- `os.system("python␣child.py")`,

или...

Многопроцессная реализация алгоритма слияния

```
1  import multiprocessing
2  import math
3
4  def parallel_merge_sort(arr: list[Number]) \
5                              -> list[Number]:
6    processes = multiprocessing.cpu_count()
7    with multiprocessing.Pool(processes = processes) \
8                                              as pool:
9      size = math.ceil(len(arr) / processes)
10     arr = [arr[i * size:(i + 1) * size]
11            for i in range(processes)]
12     arr = pool.map(merge_sort, arr)
13     while len(arr) > 1:
14       extra = arr.pop() if len(arr) % 2 == 1 else None
15       arr = [[arr[i], arr[i + 1]]
16              for i in range(0, len(arr), 2)]
17       arr = pool.map(merge, arr) + \
18             ([extra] if extra else [])
19   return arr[0]
20
```

Многопроцессная реализация алгоритма слияния. Часть 1

```
# input arr = [38, 25, 77, 45, 46, 64, 88, 76, 97, 35]

processes = multiprocessing.cpu_count() # processes=8
pool = multiprocessing.Pool(processes = processes)
size = math.ceil(len(arr) / processes) # size = 2
arr = [arr[i * size:(i + 1) * size]
       for i in range(processes)]
# arr = [[38, 25], [77, 45], [46, 64], [88, 76],
#        [97, 35], [], [], []]
arr = pool.map(merge_sort, arr) # T^{sort}(n / 2)
# arr = [[25, 38], [45, 77], [46, 64], [76, 88],
#        [35, 97], [], [], []]

```

Многопроцессная реализация алгоритма слияния. Часть 2

```
# arr = [[25, 38], [45, 77], [46, 64], [76, 88],
#        [35, 97], [], [], []]
while len(arr) > 1:
  extra = arr.pop() if len(arr) % 2 == 1 else None
  arr = [[arr[i], arr[i + 1]]
         for i in range(0, len(arr), 2)]
  # arr = [[[25, 38], [47, 77]], [[46, 64], [76, 88]],
  #        [[35, 97], []], [[], []]]
  arr = pool.map(merge, arr) + \
        ([extra] if extra else []) # T^{merge}(n)
  # Iteration 1 out: [[25, 38, 45, 77],
  #                   [46, 64, 76, 88],
  #                   [35, 97], []]
  # Iteration 2 out:
  # [[25, 38, 45, 46, 64, 76, 77, 88], [35, 97]]
  # Iteration 3 out:
  # [[25, 35, 38, 45, 46, 64, 76, 77, 88, 97]]
return arr[0]

```

Временная сложность

$$T^{sort}(n) = T^{sort}\left(\frac{n}{2}\right) + T^{merge}(n) = T^{sort}\left(\frac{n}{2}\right) + \Theta(\log n) \quad (13)$$

Код из презентации:
github.com/ValeryVerkhoturov/merge-sort-presentation