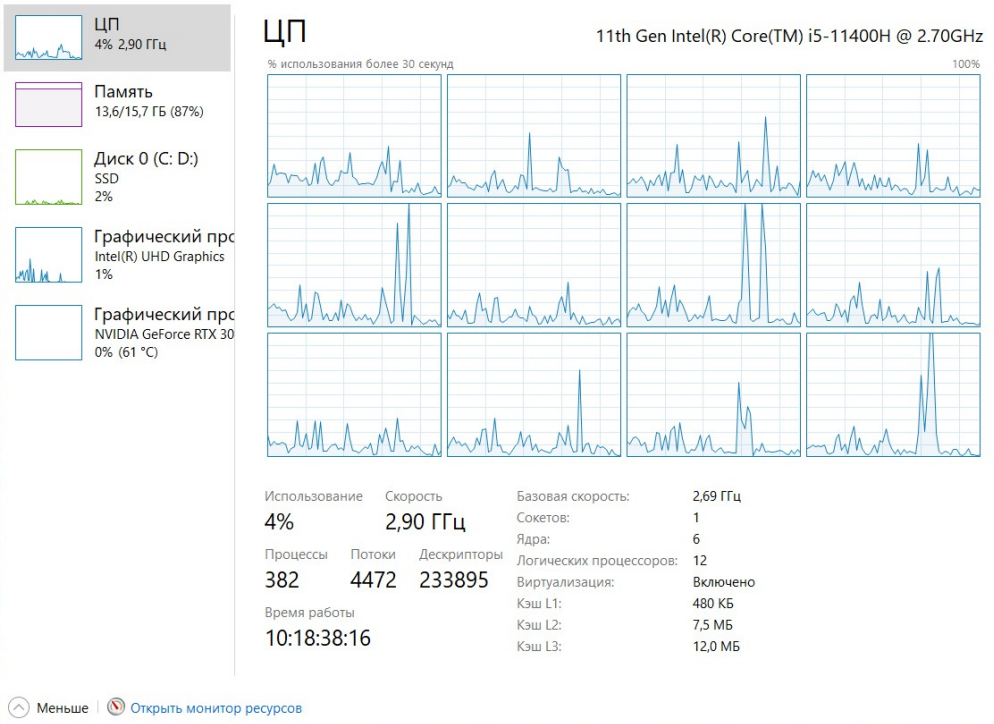
Васильева Елизавета Валерьевна АТ-03
Технологии программирования. 13 задание task_01 и task_02 с асинхронными вычислениями

TASK_02

```python
import asyncio
import time
import math

# Функции для АТ-03

async def fibonacci(n):  # содержимое функции не менять
    """Возвращает последнюю цифру n-го числа Фибоначчи."""
    if n <= 0:
        return 0
    elif n == 1:
        return 1

    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, a + b
    print(f'fibonacci = {b % 10}')
    return b % 10

async def trapezoidal_rule(f, a, b, n):  # содержимое функции не менять
    """Вычисляет определенный интеграл функции f от a до b методом трапеций с n шагами."""
    h = (b - a) / n
    integral = (f(a) + f(b)) / 2.0
    for i in range(1, n):
        integral += f(a + i * h)
    print(f'trapezoidal_rule = {integral * h}')
    return integral * h

async def main():
    start_time = time.perf_counter()
```

```python

    # Создаем задачи для асинхронного выполнения
    fib_task = asyncio.create_task(fibonacci(700003))
    trap_task = asyncio.create_task(trapezoidal_rule(math.sin, a: 0, math.pi, n: 20000000))

    # Ожидаем завершения обеих задач
    await fib_task
    await trap_task

    end_time = time.perf_counter()
    print(f'processes time: {end_time - start_time:0.2f} seconds\n')

if __name__ == '__main__':
    asyncio.run(main())
```

```
fibonacci = 7
trapezoidal_rule = 2.000000000000087
processes time: 5.17 seconds
```



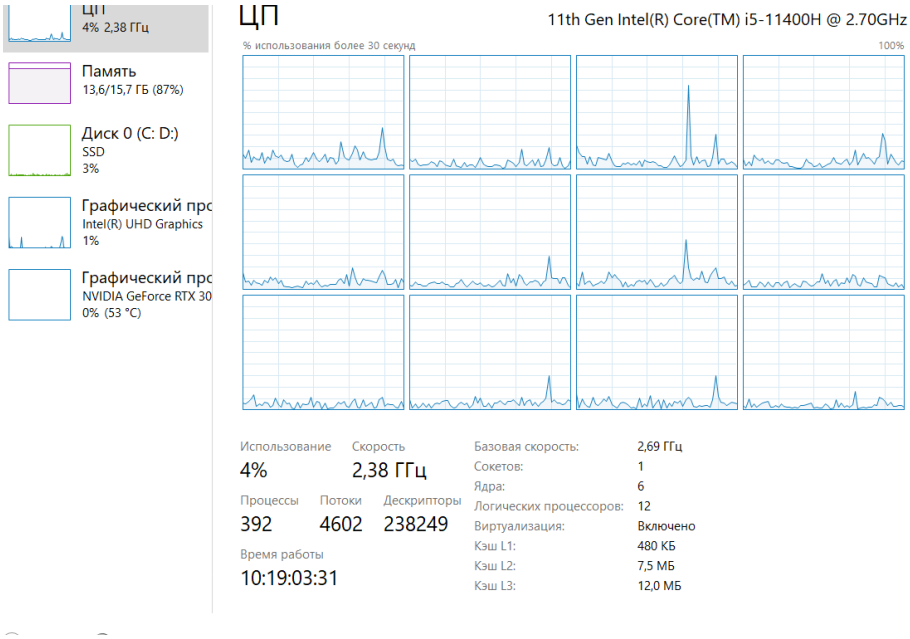
В данном случае асинхронное выполнение не дало значительного преимущества, поскольку задачи являются CPU-bound(скорость выполнения зависит преимущественно от мощности процессора, например, арифметические операции). Асинхронный подход лучше всего подходит для I/O-bound задач (IO bound задачи -это задачи, совокупное время выполнения которых в основном определяется временем выполнения всех операций ввода/вывода.), где время ожидания может быть использовано для выполнения других операций. В данном случае, поскольку оба вычисления требуют значительных ресурсов процессора, асинхронное выполнение не смогло показать преимущества по сравнению с другими методами.

TASK_01

```python
import asyncio
import aiohttp
import time

# список url
urls = ['https://www.example.com'] * 10

async def fetch_url(session, url):
    async with session.get(url) as response:
        return await response.text()

async def async_fetch_all():
    async with aiohttp.ClientSession() as session:
        tasks = [fetch_url(session, url) for url in urls]
        return await asyncio.gather(*tasks)

def async_execution():
    start_time = time.perf_counter()  # время старта
    asyncio.run(async_fetch_all())  # выполнение асинхронных задач
    end_time = time.perf_counter()  # время окончания
    print(f'processes time: {end_time - start_time:0.2f} seconds\n')

if __name__ == '__main__':
    async_execution()
```

```
processes time: 1.07 seconds
```

Для выполнения HTTP-запросов потребуется дополнительная библиотека, т. к. библиотека **requests** является синхронной, её нельзя использовать напрямую в асинхронном контексте. **Aiohttp** предназначена для асинхронного выполнения запросов.



Когда одна задача ожидает ответа от сервера, цикл событий может переключиться на выполнение другой задачи. Это значительно сокращает общее время выполнения по сравнению с последовательным подходом. Но наиболее эффективным способом, всё также остается выполнение через потоки.