Лабораторная работа 6

Ильин Вячеслав, М3236

На рисунках 1 и 2 представлены параметры виртуальной машины и хостового компьютера, на которых проводился эксперимент.

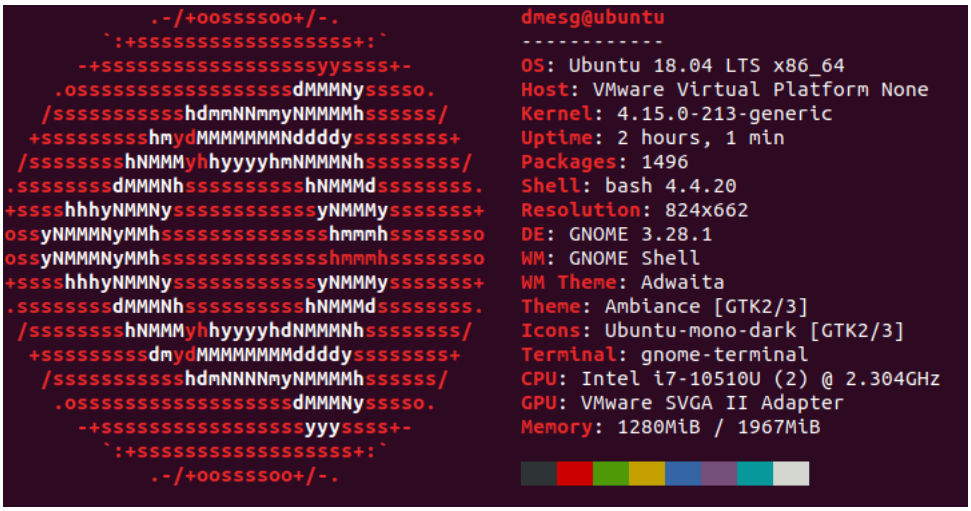

Рисунок 1 – параметры виртуальной машины

```
Имя узла:                     LAPTOP-EKDLF31N
Название ОС:                  Майкрософт Windows 11 Домашняя для одного языка
Версия ОС:                    10.0.22621 Н/Д построение 22621
Изготовитель ОС:              Microsoft Corporation
Параметры ОС:                 Изолированная рабочая станция
Сборка ОС:                    Multiprocessor Free
Зарегистрированный владелец:  slavailyin2004@gmail.com
Зарегистрированная организация: Н/Д
Код продукта:                 00342-41402-99782-AAOEM
Дата установки:               28.01.2023, 22:38:02
Время загрузки системы:       22.12.2023, 21:29:50
Изготовитель системы:         LENOVO
Модель системы:               81NF
Тип системы:                  x64-based PC
Процессор(ы):                 Число процессоров - 1.
                              [01]: Intel64 Family 6 Model 142 Stepping 12 GenuineIntel ~1803 МГц
Версия BIOS:                  LENOVO CKCN11WW(V1.01), 19.07.2019
Папка Windows:                C:\WINDOWS
Системная папка:              C:\WINDOWS\system32
Устройство загрузки:          \Device\HarddiskVolume1
Язык системы:                 ru;Русский
Язык ввода:                   ru;Русский
Часовой пояс:                 (UTC+03:00) Москва, Санкт-Петербург
Полный объем физической памяти:   7 991 МБ
Доступная физическая память:      750 МБ
Виртуальная память: Макс. размер: 23 863 МБ
Виртуальная память: Доступна:     5 670 МБ
Виртуальная память: Используется: 18 193 МБ
Расположение файла подкачки:      C:\pagefile.sys
```

Рисунок 2 – параметры хостового компьютера

1. Эксперимент с выполнением вычислительно сложной задачи

Я написал функцию, которая находит сумму простых чисел, меньших N – входного параметра. Код представлен на рисунке 1. Я взял N = 20000, при нем программа работает ~3 секунды.

```bash
#!/bin/bash

is_prime() {
    local number=$1
    for (( i=2; i*i<=number; i++ )); do
        if (( number % i == 0 )); then
            return 1
        fi
    done
    return 0
}

sum_of_primes() {
    local limit=$1
    local sum=0
    local number=2

    while (( number <= limit )); do
        if is_prime $number; then
            sum=$((sum + number))
        fi
        ((number++))
    done

    echo $sum
}
```

Рисунок 2 – код программы, считающей сумму простых чисел до N

```bash
#!/bin/bash

script="./seq_run.sh"
for N in {1..20}; do
total=0
for i in {1..10}; do
time_str=$( { time bash $script $N; } 2>&1 | grep real | awk '{print $2}')
real_time=$(echo $time_str | awk '{split($1,a,"m"); split(a[2],s,"s"); print (a[1]*60) + s[1]}')
total=$(echo "$total + $real_time" | bc)
done
echo "$total"
done
```

Рисунок 2 – код программы, отслеживающей время исполнения

1.1. 1 процессор

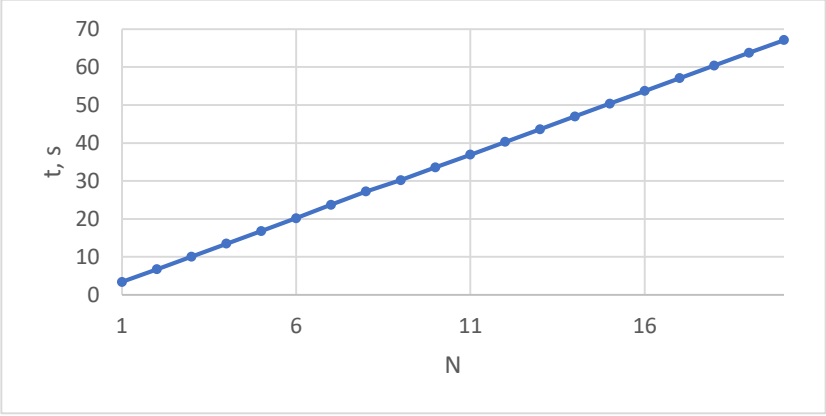

График 3 – зависимость времени работы программы от количества запусков при последовательном исполнении

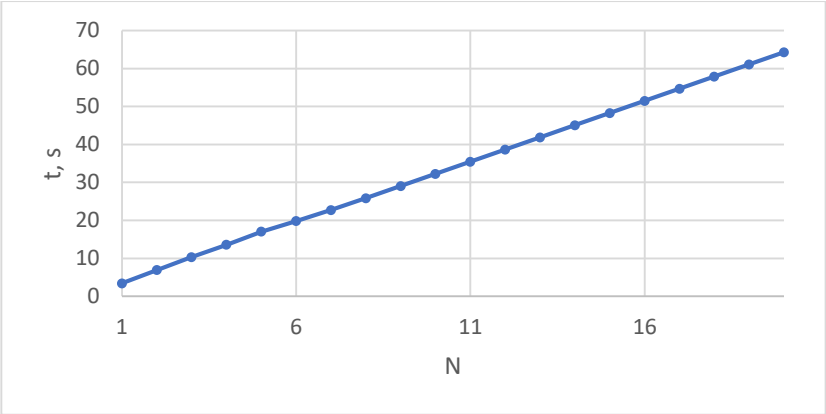

График 2 – зависимость времени работы программы от количества запусков при параллельном исполнении

Оба графика – линейные. Это объясняется тем, что программы выполняются на одном процессоре, поэтому даже при параллельной работе скрипты будут запускаться последовательно. Время работы прямо пропорционально количеству запусков программы.

1.2. 2 процессора

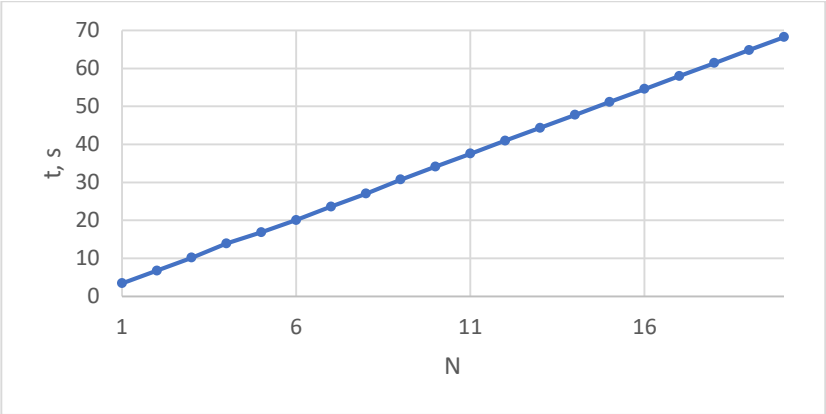

График 3 – зависимость времени работы программы от количества запусков при последовательном исполнении

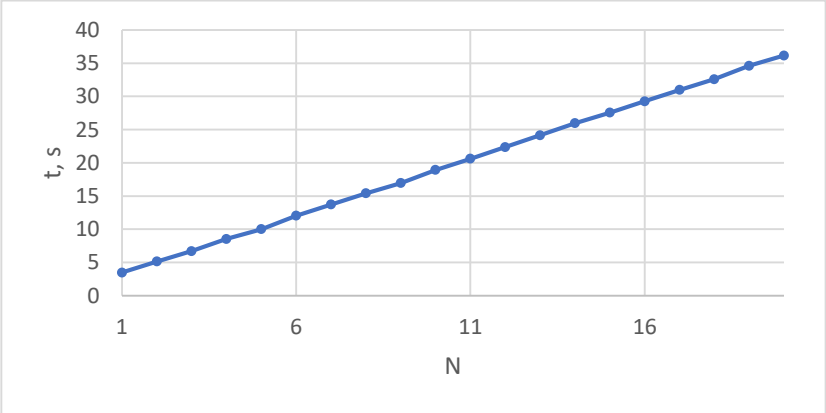

График 4 – зависимость времени работы программы от количества запусков при параллельном исполнении

График 3 аналогичен первым двум. Так как скрипты запускаются последовательно, то количество процессоров не влияет на общее время исполнения. На графике 4 наклон прямой уменьшился примерно в 2 раза, по сравнению с предыдущими результатами. Это объясняется тем, что при параллельной работе скриптов, они распределяются между двумя процессорами и, так как скрипты работают в среднем одинаковое время, они заканчиваются так же примерно одновременно. Таким образом, программы исполняются в 2 раза быстрее.

2. Эксперимент с выполнением задач с большими объемами считываемых и сохраняемых данных

Для второго эксперимента я создавал файлы размера 100кб, так что обработка занимала ~3 секунды. На рисунке 3 представлен код этой программы. На рисунке 4 представлен код скрипта, обрабатывающего эти файлы. Он читает последовательно числа, умножает на 2 и записывает в конец файла, каждый раз сохраняя результат.

```
#!/bin/bash

head -c 102400 /dev/urandom | od -An -td4 -w4 > "$1"
```

Рисунок 3 – код программы, генерирующей файл с числами

```
#!/bin/bash

file_name="$1"
line_count=$(wc -l < "$file_name")
current_line=0

while IFS= read -r number && (( current_line < line_count )); do
    ((current_line++))
    echo $((number * 2)) >> "$file_name"
done < "$file_name"
```

Рисунок 4 – код программы, выполняющий операции с файлом

2.1. 1 процессор

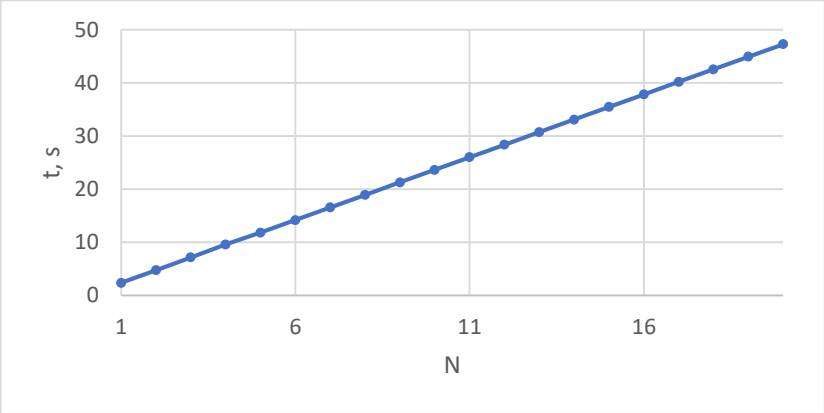

График 5 – зависимость времени работы программы от количества запусков при последовательном исполнении

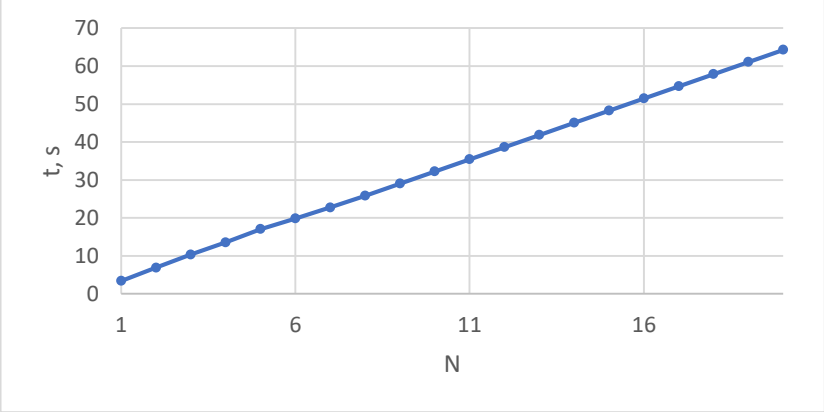

График 6 – зависимость времени работы программы от количества запусков при параллельном исполнении

2.2. 2 процессора

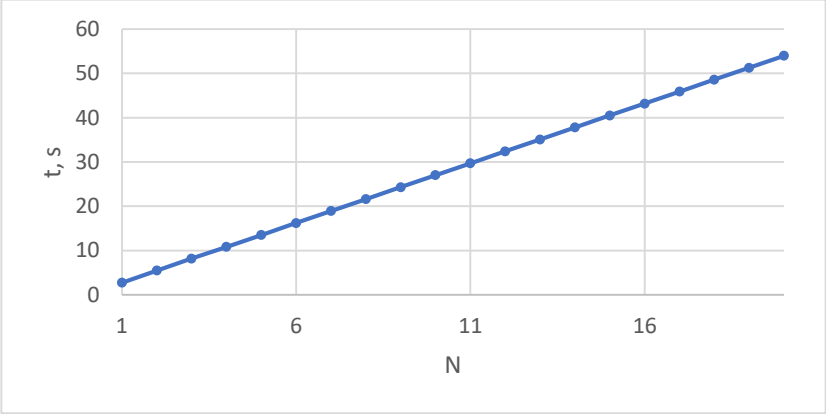

График 7 – зависимость времени работы программы от количества запусков при последовательном исполнении

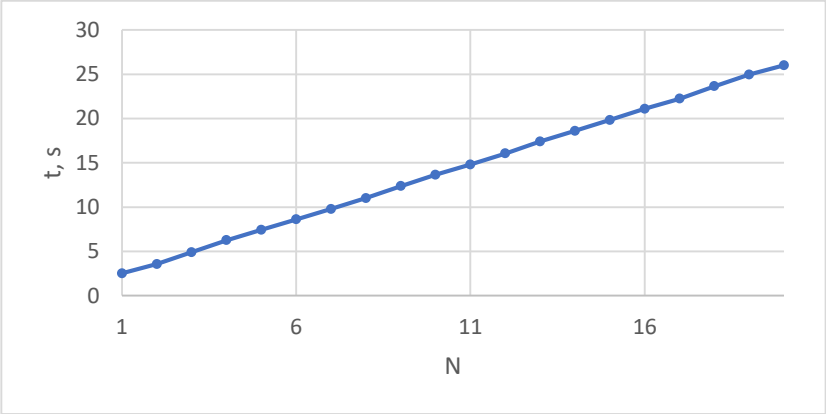

График 8 – зависимость времени работы программы от количества запусков при параллельном исполнении

Результаты получились аналогичные с первым экспериментом. Все графики линейные, первые 3 почти не различаются, но при параллельном запуске на 2 процессорах время выполнения уменьшилось примерно в два раза, как и в первом эксперименте.