Отчёт по лабораторной работе номер 6.

Параметры из вывода команды **top**:

```
MiB Mem :   1826.8 total,   1356.4 free,    183.5 used,    286.9 buff/cache
MiB Swap:    820.0 total,    820.0 free,      0.0 used.   1487.4 avail Mem
```

Параметры виртуальной машины:

Memory	2 GB
Processors	2
Hard Disk (SATA)	8 GB
Network Adapter	NAT
USB Controller	Present
Display	Auto detect

Алгоритмы:

В качестве первого алгоритма я взял вычисление странной функции в R^3:

```
ld function(ld x, ld y, ld z) {
    int n = 10000000*2.5;
    for (int i = 0; i < n; i++) {
        ld new_x = sin(x*y - 2*y + z) + rand()%10;
        ld new_y = cos(x - 3*y*z + 4*z) + rand()%10;
        ld new_z = sin(5*x + y - z*y) + rand()%10;
        x = new_x;
        y = new_y;
        z = new_z;
    }
    return (x + y + z) / 2;
}
```

Второй алгоритм для работы с файлами:

```
#include <fstream>
#include <iostream>
#include <string>

int main() {
    std::string file_name;
    std::cin >> file_name;

    std::ifstream in("files/" + file_name);
    std::ifstream IN("config");

    int num_of_numbers;
    IN >> num_of_numbers;
    IN.close();

    std::ofstream out("files/" + file_name, std::ios::app);
    for (int i = 0; i < num_of_numbers; i++) {
        int x;
        in >> x;
        out << 2 * x << " ";
    }

    in.close();
    out.close();

    std::cout << "done" << std::endl;

    return 0;
}
```

Скрипт для подсчёта средних значений:

```
for ((n=1; n <= 20; n++))
do
    sum=0
    for ((i=1; i <= 10; i++))
    do
        res=`{ \time -f "%e" ./launch_1.sh $n ; } 2>&1 1>/dev/null`
        sum=$(echo $sum $res | awk '{print $1 + $2}')
    done
    sum=$(echo $sum 10 | awk '{print $1 / $2}')
    echo "$sum"
done
```

Пример запускающего скрипта:

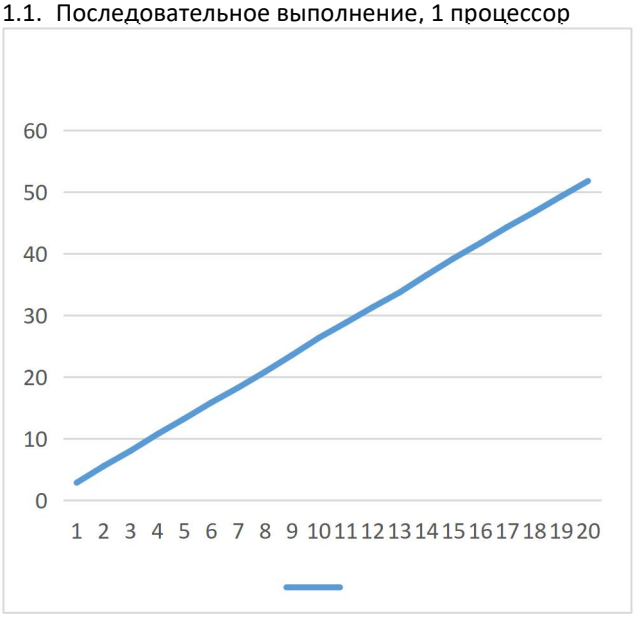
```
#!/bin/bash

: > ./ready.txt
for ((i=1; i <= $1; i++))
do
    echo "file_$i" | ./task_2 >> ./ready.txt &
done

wait
```

1. Группа экспериментов номер 1

1.1. Последовательное выполнение, 1 процессор



Мы видим линейную зависимость, т.к. у нас один процессор и послед. выполнение.

```
%Cpu(s):100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   7848.6 total,   7538.9 free,    217.8 used,     91.9 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   7455.8 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
   92 root      20   0    5884   1520   1364 R  99.9   0.0   0:00.88 task_1
```

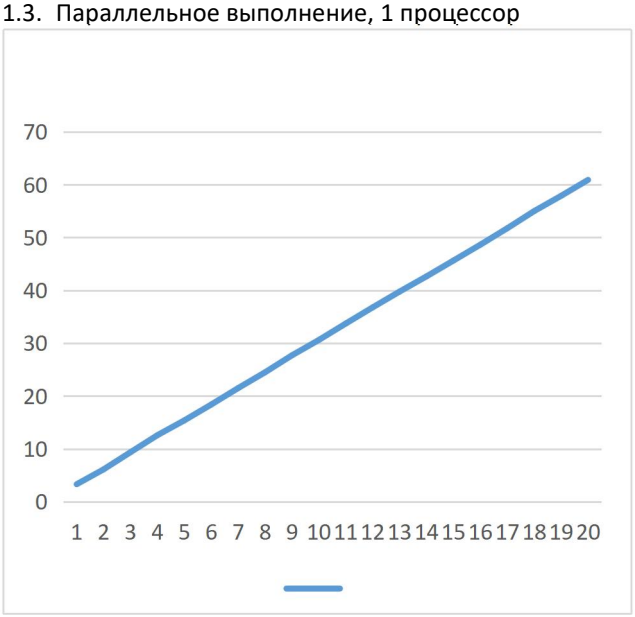
На этом скрине мы видим, что память на нуле, т.к. мы её вообще не используем, cpu загружен на 100%.

1.2. Последовательное выполнение, 2 процессора

Аналогично лин. зависимость, всё выполняется последовательно, поэтому общее время выполнения такое же.

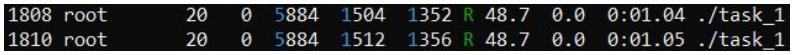
Здесь мы видим, что приоритет отдаётся одному из процессоров (бывает, что и второму), оба не используюся, т.к. запуск послед.
Аналогично пункту 1.2 главная задача - это task_1, память на нуле.

1.3. Параллельное выполнение, 1 процессор



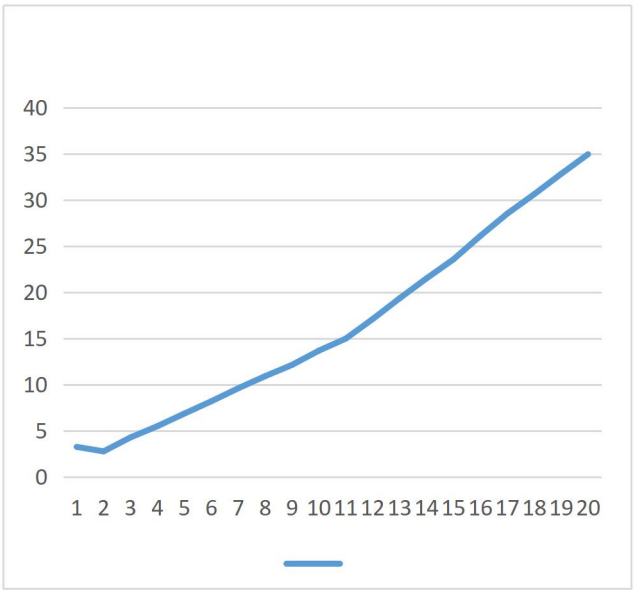
Хоть выполнение и параллельное, процессор то 1, поэтому не видно никакого эффекта.

Единственный процессор загружен на макс.



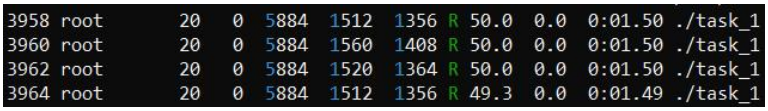
Ресурсы распр. справедливо.

1.4. Параллельное выполнение, 2 процессора



Этот случай интереснее. На 1 запуске мы потратили ~3 секунды, на 2 запуске у нас задачи работали параллельно на двух процессорах, поэтому время тоже ~3 секунды, даже чуть меньше. Далее мы наблюдаем линейный рост, но коэффициент стал в 2 раза меньше, т.к. у нас два проц.

Здесь, в отличие от п.1.2., оба процессора загружены полностью.

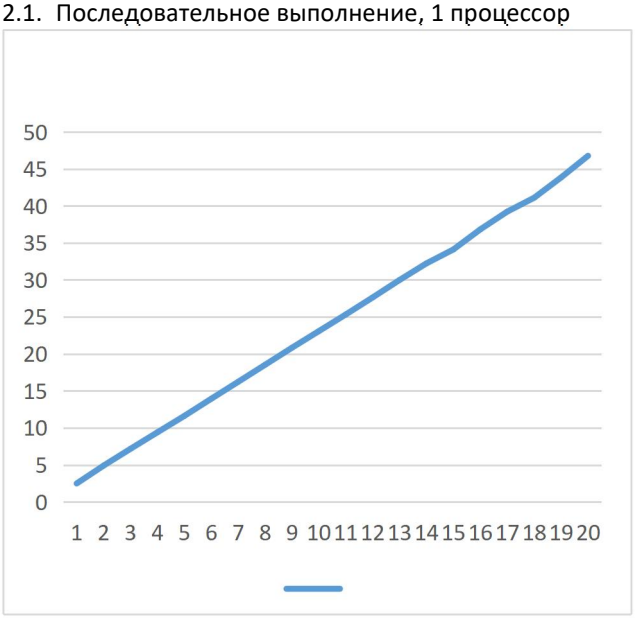


Распределение ресурсов справедливое.

2. Группа экспериментов номер 2

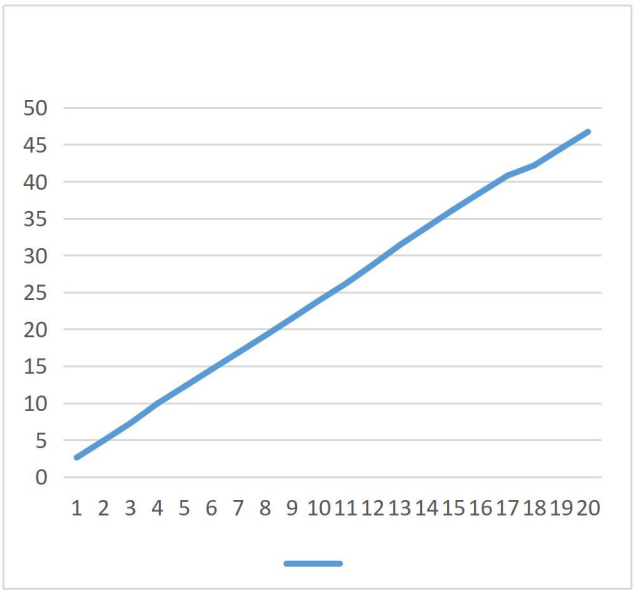
У меня при N=4E6 программа работает 2-3 секунды.

2.1. Последовательное выполнение, 1 процессор



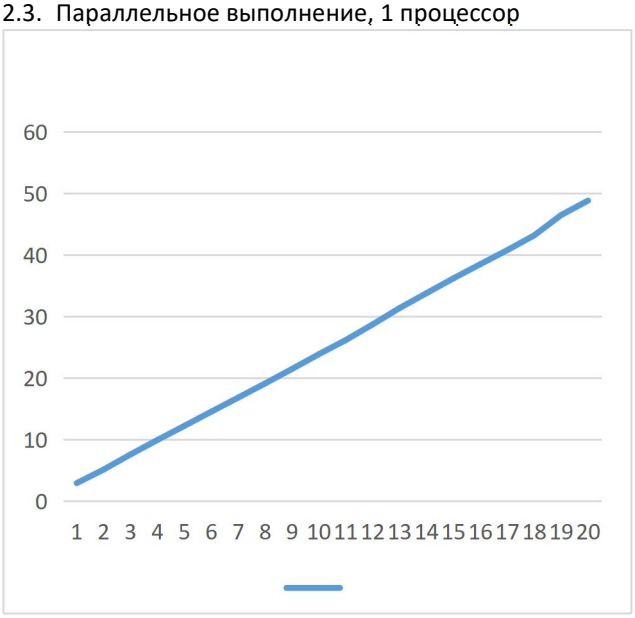
Аналогично имеем лин. рост.

2.2. Последовательное выполнение, 2 процессора



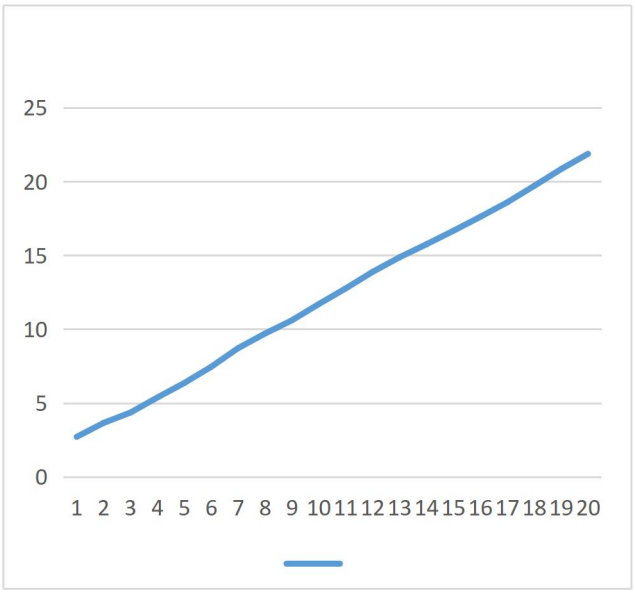
Выполнение послед., процессора 2, но меняться ничего не должно, поэтому мы видим лин. завис. с таким же углом.

2.3. Параллельное выполнение, 1 процессор



Опять же ничего не меняется, всё та же верхняя граница, так же лин. зависимость.

2.4. Параллельное выполнение, 2 процессора



Стало 2 процессора, поэтому программа стала работать в 2 раза быстрее.

В последних 4х экспериментах cpu загружены не полностью, т.к. основная работа идёт с памятью.

Вывод: если задача не распр., то не важно сколько у нас процессоров.