Отчёт по лабораторной работе №6 (вариант повышенной сложности). Экспериментальная оценка параметров производительности операционной системы

Кирьяк Александр Александрович, M3234

Лабораторная выполнялась на ноутбуке с процессором 6 ядер и 12 потоков, 3.30 GHz.

```
MiB Mem :   6863.1 total,   6343.7 free,    301.5 used,    218.0 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   6341.0 avail Mem
```

Эксперименты с последовательным и параллельным выполнением вычислительно сложных задач

В качестве сложного процессозатратного алгоритма выбрано вычисление синуса в заданной точке с высокой точностью. Для этого воспользовались рядом Маклорена для синуса:

$$\sin x = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{x^{2k-1}}{(2k-1)!}, \quad x \in \mathbb{R}.$$

Программа на C++ выглядит следующим образом:

sine_calculating.cpp

```cpp
#include <iostream>

using namespace std;

double sin(int x, int steps = 10) {
    double y = x;
    double quotient = x;

    for (int k = 2; k <= steps; k++) {
        quotient *= (double) x * x / (2 * (k - 1)) / (2 * k - 1);
        if (k % 2 == 0) {
            y -= quotient;
        } else {
            y += quotient;
        }
    }

    return y;
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        throw runtime_error("Sine argument expected");
    }

    int x = atoi( Str: argv[1]);
    cout << sin(x,  steps: 700'000'000) << '\n';
}
```

Количество итераций при вычислении выбрано довольно большим (700 миллионов) для того, чтобы программа работала около 2 секунд при любом заданном аргументе.

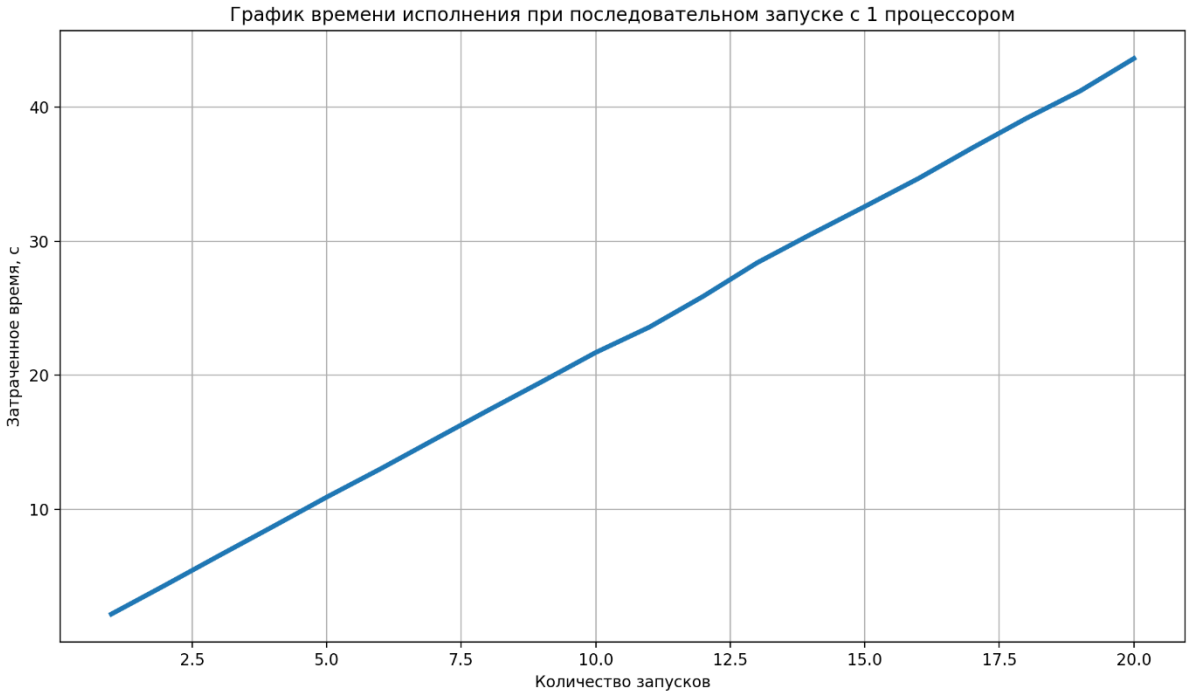
1. Последовательное вычисление с 1 процессором

```bash
#!/bin/bash

runner="seq_calc_runner.sh"
compile_name="sine_calculating"
g++ "$compile_name.cpp" -o $compile_name

log_file="seq_calc.log"
if [[ ! -f $log_file ]]
then
    touch $log_file
else
    echo -n > $log_file
fi

MAX_N=20
runs=10

for ((N = 1; N <= $MAX_N; N++))
do
    echo "$N:" >> $log_file
    for ((i = 1; i <= $runs; i++))
    do
        \time -f "%E" "./$runner" $N > /dev/null 2>>$log_file
    done
done
```

seq_calc.sh

```bash
#!/bin/bash

run_name="sine_calculating"

for ((x = 1; x <= $1; x++))
do
    ./$run_name $x
done
```

seq_calc_runner.sh



Поскольку вычисления производились последовательно на одном процессоре, функция получилась почти линейная.

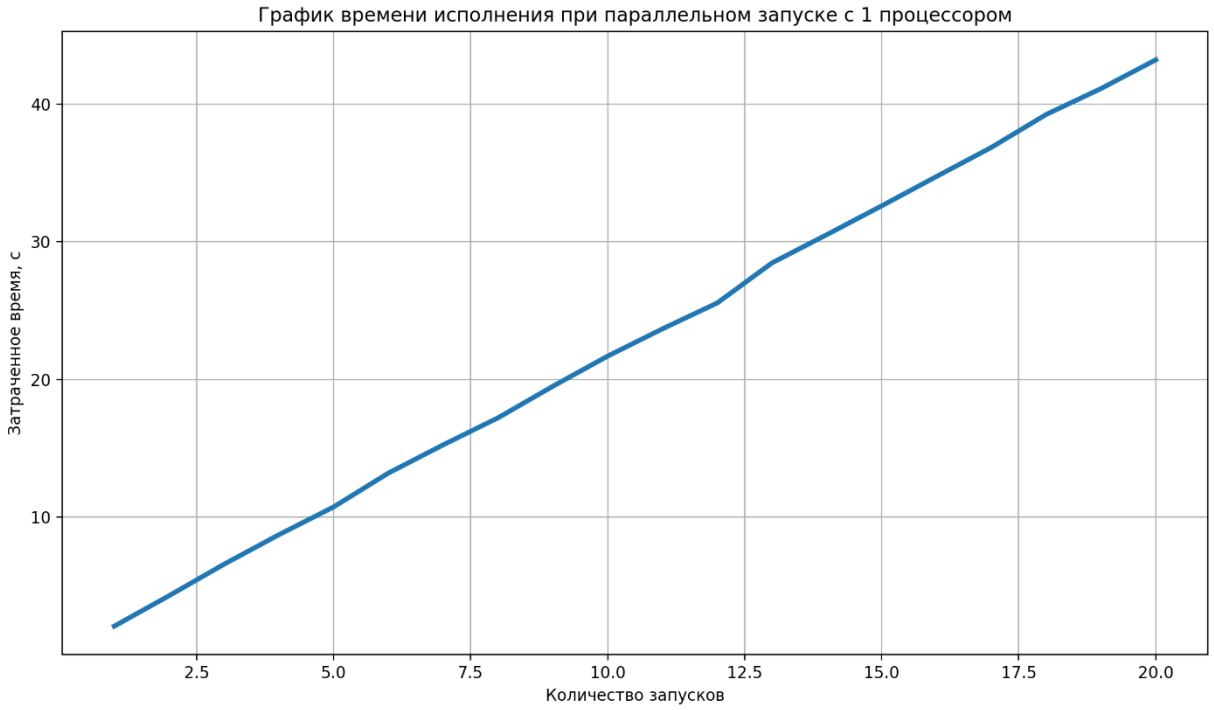
2. Параллельное вычисление с 1 процессором

```bash
#!/bin/bash

runner="parallel_calc_runner.sh"
compile_name="sine_calculating"
g++ "$compile_name.cpp" -o $compile_name

log_file="parallel_calc.log"
if [[ ! -f $log_file ]]
then
    touch $log_file
else
    echo -n > $log_file
fi

MAX_N=20
runs=10

for ((N = 1; N <= $MAX_N; N++))
do
    echo "$N:" >> $log_file
    for ((i = 1; i <= $runs; i++))
    do
        \time -f "%E" "./$runner" $N > /dev/null 2>>$log_file
    done
done
```

parallel_calc.sh

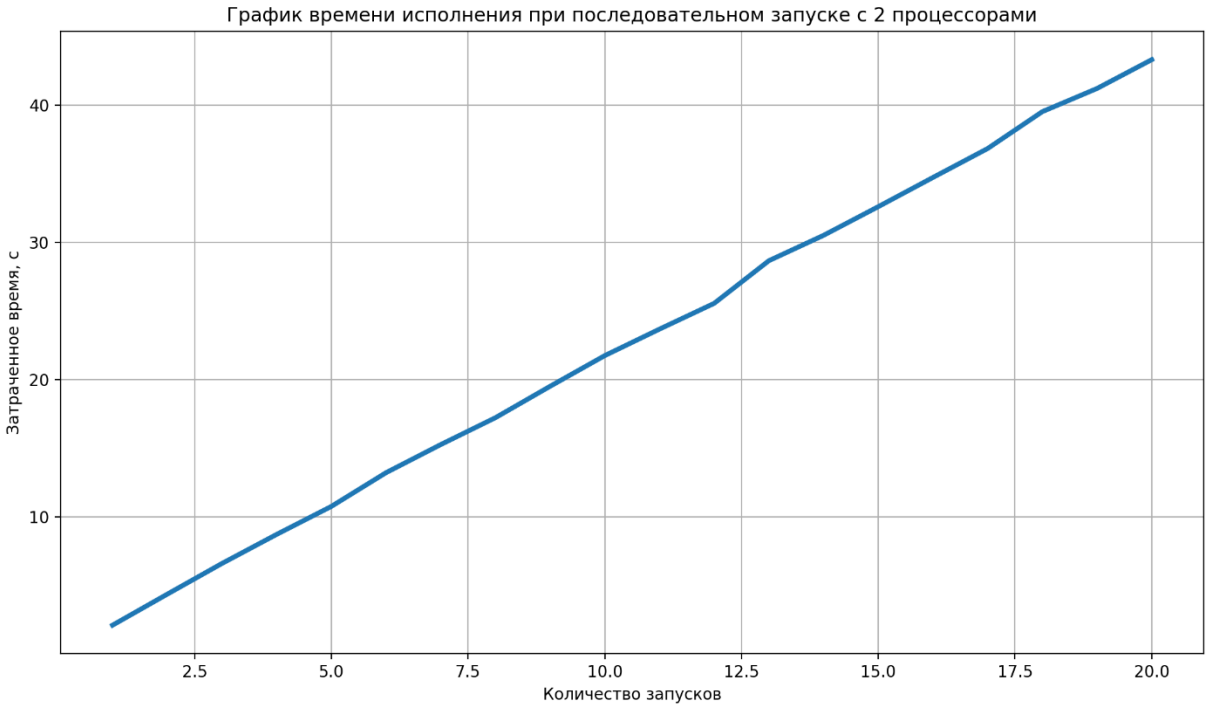
```bash
#!/bin/bash

run_name="sine_calculating"

for ((x = 1; x <= $1; x++))
do
    ./$run_name $x &
done
wait
```

parallel_calc_runner.sh



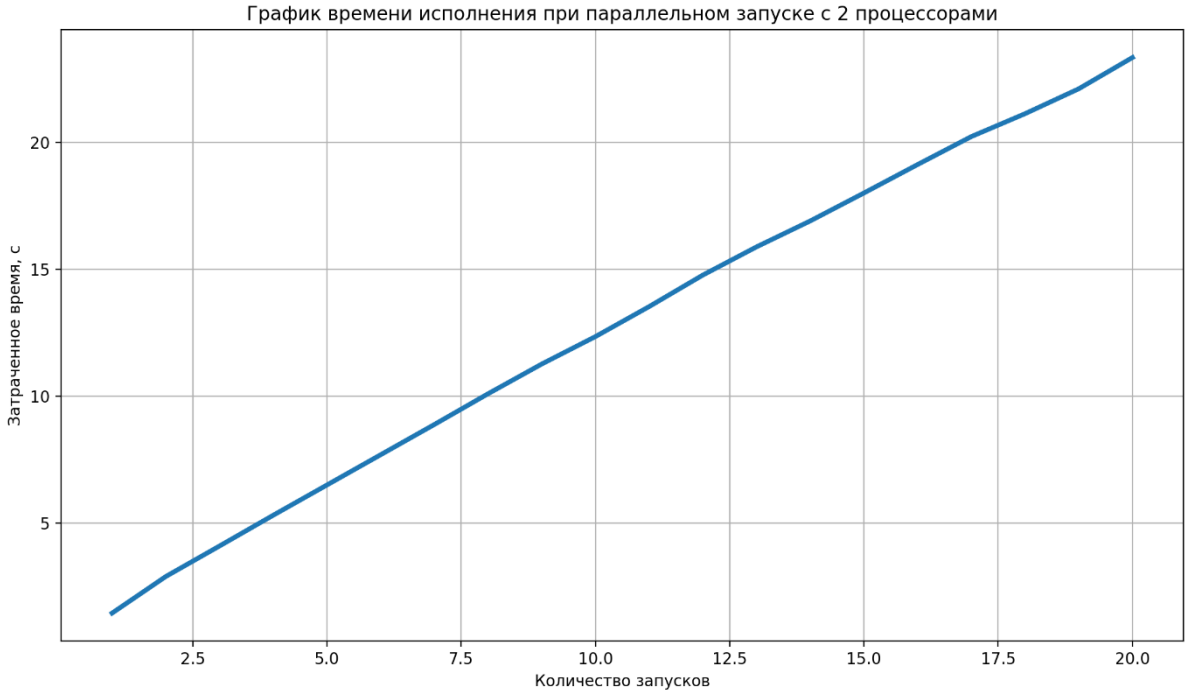
Несмотря на то, что запуски делались параллельно, это не дало преимущества. Это связано с тем, что использовался только 1 процессор.

3. Последовательное вычисление с 2 процессорами



Так как вычисления запускались последовательно, 2 процессора не принесли особых ускорений в скорости работы.

4. Параллельное вычисление с 2 процессорами



В этом случае получилось ускорение примерно в 2 раза. Это произошло благодаря второму процессору: теперь планировщик эффективно распараллеливает работу процессов.

Эксперимент с параллельным и последовательным выполнением задач с большими объемами считываемых и сохраняемых данных

Предварительно были созданы 20 файлов, состоящие из случайных чисел в диапазоне [-10^9; 10^9]. Все файлы содержали 4’300’000 чисел, чтобы обработка каждого файла занимала порядка 2 секунд. Программа, выполняющая преобразование, согласно заданию:

file_changing.cpp

```cpp
#include <iostream>
#include <fstream>

using namespace std;

const int COUNT_NUMBERS = 4'300'000;

int main(int argc, char* argv[]) {
    if (argc != 2) {
        throw runtime_error("File number expected");
    }

    string file = "file" + string( s: argv[1]);
    ifstream input( s: file);
    ofstream output( s: file, mode: fstream::app);

    for (int i = 0; i < COUNT_NUMBERS; i++) {
        int num;
        input >> num;
        output << num * 2 << ' ';
    }
}
```

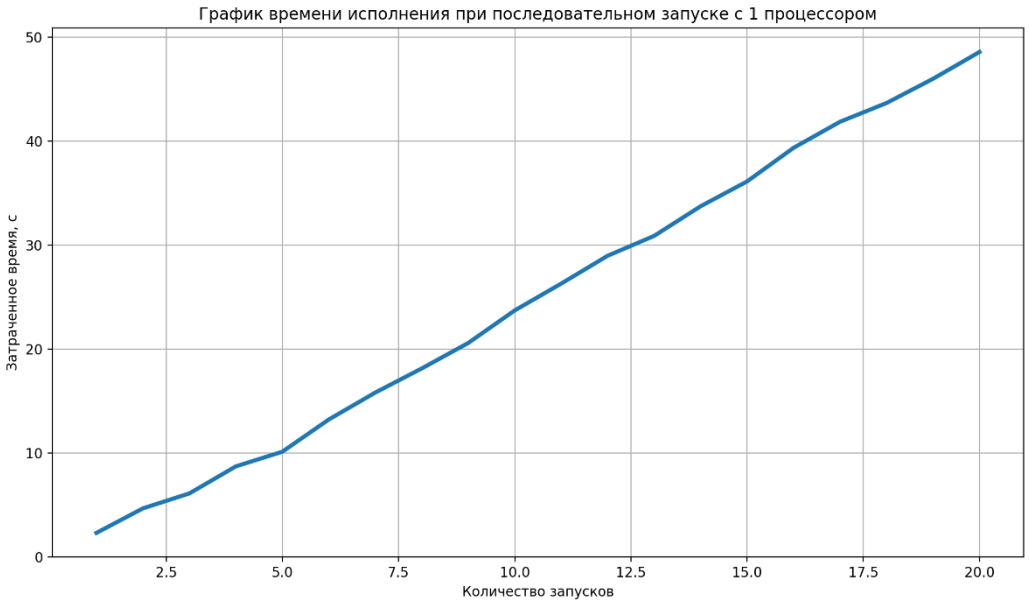
1. Последовательная обработка с 1 процессором

```bash
#!/bin/bash

runner="seq_mem_runner.sh"
compile_name="file_changing"
g++ "$compile_name.cpp" -o $compile_name

log_file="seq_mem.log"
if [[ ! -f $log_file ]]
then
    touch $log_file
else
    echo -n > $log_file
fi

MAX_N=20
runs=10

for ((N = 1; N <= $MAX_N; N++))
do
    echo "$N:" >> $log_file
    for ((i = 1; i <= $runs; i++))
    do
        \time -f "%E" "./$runner" $N > /dev/null 2>>$log_file
    done
done
```

seq_mem.sh

```bash
#!/bin/bash

run_name="file_changing"

for ((no = 1; no <= $1; no++))
do
    ./$run_name $no
done
```

seq_mem_runner.sh



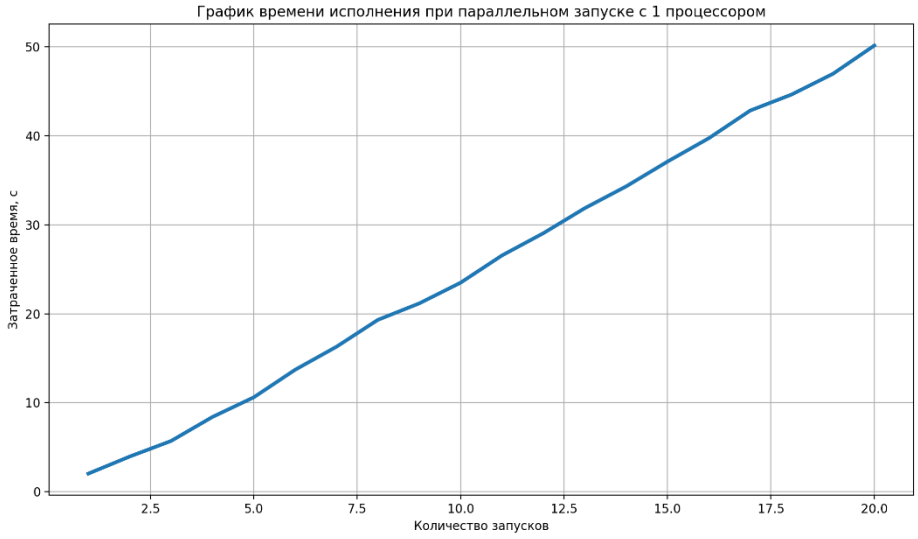
2. Параллельная обработка с 1 процессором

```bash
#!/bin/bash

runner="parallel_mem_runner.sh"
compile_name="file_changing"
g++ "$compile_name.cpp" -o $compile_name

log_file="parallel_mem.log"
if [[ ! -f $log_file ]]
then
    touch $log_file
else
    echo -n > $log_file
fi

MAX_N=20
runs=10

for ((N = 1; N <= $MAX_N; N++))
do
    echo "$N:" >> $log_file
    for ((i = 1; i <= $runs; i++))
    do
        \time -f "%E" "./$runner" $N > /dev/null 2>>$log_file
    done
done
```

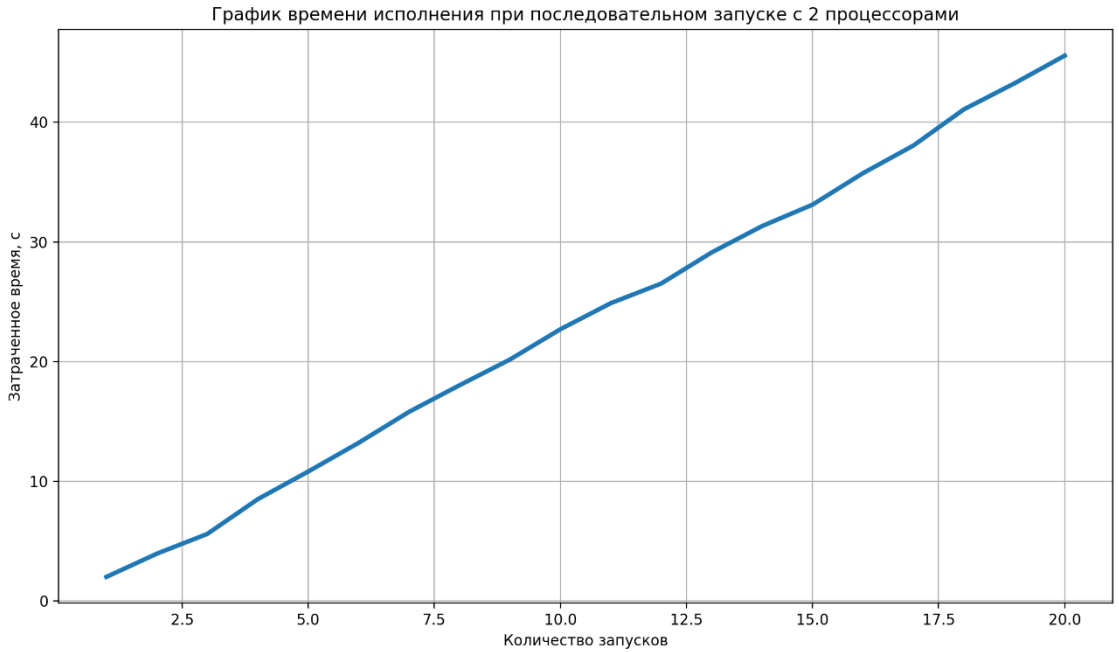
parallel_mem.sh

```bash
#!/bin/bash

run_name="file_changing"

for ((no = 1; no <= $1; no++))
do
    ./$run_name $no &
done
wait
```

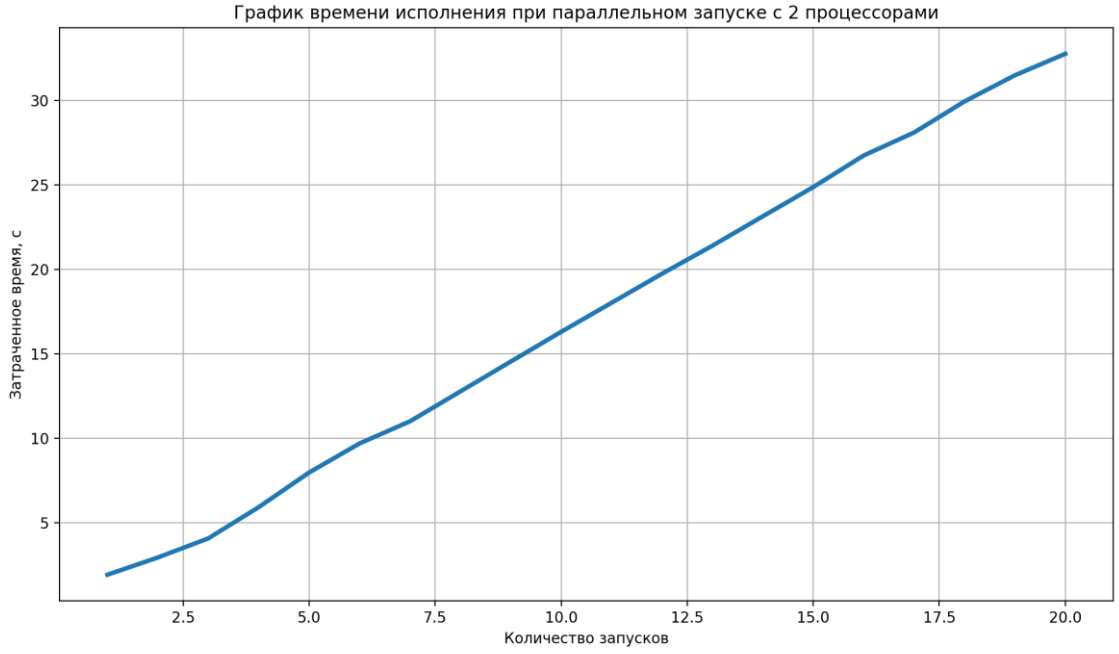
parallel_mem_runner.sh



3. Последовательная обработка с 2 процессорами



4. Параллельная обработка с 2 процессорами



Результаты получились схожие с первым экспериментом. Лучшую производительность показывает параллельный запуск с 2 процессорами. Выигрыш примерно в 2 раза по сравнению с другими видами обработки.