**Экспериментальная оценка параметров производительности операционной системы**

Параметры компьютера:

Оперативная память – 16 Гб

CPU cores – 6

Параметры виртуальной машины:

Оперативная память – 4 Гб

CPU cores – 1/2

**1. Эксперименты с последовательным и параллельным выполнением вычислительно сложных задач**

Вычислительно сложный алгоритм

Входные данные: координаты точки x, y, z

Выходные данные: значение интеграла функции в заданной точке методом Монте-Карло

#!/bin/bash

function f {

    local x=$1

    local y=$2

    local z=$3

    echo "$x^2 + $y^2 + $z^2" | bc -l

}

function rand {

    local min=$1

    local max=$2

    echo "$(awk -v min=$min -v max=$max 'BEGIN{srand(); print min + rand() \* (max - min)}')"

}

function monte\_carlo\_integral {

    local num\_count=120

    local x=$1

    local y=$2

    local z=$3

    local sum=0

    for ((i=0; i<num\_count; i++)); do

        local rx=$(rand $(echo "$x-1" | bc) $(echo "$x+1" | bc))

        local ry=$(rand $(echo "$y-1" | bc) $(echo "$y+1" | bc))

        local rz=$(rand $(echo "$z-1" | bc) $(echo "$z+1" | bc))

        sum=$(echo "$sum + $(f $rx $ry $rz)" | bc -l)

    done

    local volume=$(echo "2^3" | bc -l)

    local integral=$(echo "$sum / $num\_count \* $volume" | bc -l)

    echo $integral

}

x=$1

y=$2

z=$3

integral=$(monte\_carlo\_integral $x $y $z)

1. Первый эксперимент

a. Используем один процессор

b. Скрипт, который, получив параметр N, запускает последовательно друг за другом N вычислений для разных значений входных параметров (значение каждой координаты от 1 до 100)

#!/bin/bash

n=$1

for ((i=0; i<$n; i++)); do

    x=$((RANDOM % 100 + 1))

    y=$((RANDOM % 100 + 1))

    z=$((RANDOM % 100 + 1))

    ./algorithm.sh $x $y $z

done

c. Скрипт, который для каждого N в диапазоне от 1 до 20 запускает 10 раз предыдущий скрипт через утилиту time и записывает в файл время, затраченное на полное выполнение данного скрипта. На выходе получается 20 серий по 10 значений

#!/bin/bash

> time.txt

for ((i=1; i<21; i++)); do

    echo "N = $i" >> time.txt

    total\_time=0

    for ((j=0; j<10; j++)) do

        exec\_time=$({ time ./1-1-b.sh $i; } 2>&1)

        echo "$exec\_time" >> time.txt

        runtime=$( echo "$exec\_time" | grep "real" | awk '{print $2}' | awk -F 'm' '{print $1\*60s$2}' | awk -F 's' '{print $1+$2}' )

        total\_time=$(echo "$total\_time + $runtime" | bc)

    done

    average\_time=$(echo "scale=3; $total\_time / 10" | bc)

    echo "Avg time = $average\_time" >> time.txt

done

d. График зависимости среднего значения времени выполнения от N

2. Второй эксперимент

a. Скрипт, который, получив параметр N, запускает параллельно N вычислений для разных значений входных параметров (значение каждой координаты от 1 до 100)

#!/bin/bash

n=$1

for ((i=0; i<$n; i++)); do

    x=$((RANDOM % 100 + 1))

    y=$((RANDOM % 100 + 1))

    z=$((RANDOM % 100 + 1))

    ./algorithm.sh $x $y $z &

done

wait

b. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

c. График зависимости среднего значения времени выполнения от N

3. Третий эксперимент

a. Используем два процессора

b. Используем скрипт, который запускает последовательно друг за другом N вычислений для разных значений входных параметров из пункта 1.b

c. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

d. График зависимости среднего значения времени выполнения от N

4. Четвертый эксперимент

a. Используем скрипт, который запускает параллельно друг за другом N вычислений для разных значений входных параметров из пункта 1.b

b. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

c. График зависимости среднего значения времени выполнения от N

**2. Эксперименты с параллельным и последовательным выполнением задач с большими объемами считываемых и сохраняемых данных**

Алгоритм, который считывает значения из файла, умножает на два и дописывает в конец этого файла. Каждое значение считывается и записывается по очереди. Подобрали размер файла – 1 МБ, чтобы обработка одного файла занимала 2-3 секунды

#!/bin/bash

process\_file() {

  local filename=$1

  local num\_lines=$(wc -l < "$filename")

  local c=1

  while [ $c -ne $num\_lines ]; do

    read number

    result=$(($number \* 2))

    echo $result >> "$filename"

    ((c++))

  done < "$filename"

}

process\_file $1

1. Первый эксперимент

a. Используем один процессор

b. Скрипт, который, получив параметр N, запускает последовательно друг за другом N вычислений для разных значений входных параметров (файлов)

#!/bin/bash

n=$1

num=$2

for ((i=1; i<=$n; i++)); do

    ./algorithm2.sh $num/file\_$i.txt

done

c. Скрипт, который для каждого N в диапазоне от 1 до 20 запускает 10 раз предыдущий скрипт через утилиту time и записывает в файл время, затраченное на полное выполнение данного скрипта. На выходе получается 20 серий по 10 значений

#!/bin/bash

> time5.txt

for ((i=1; i<21; i++)); do

    echo "N = $i" >> time5.txt

    exec\_time=$({ time ./2-1-b.sh $i test\_files\_$i; } 2>&1)

    echo "$exec\_time" >> time5.txt

    rm -rf test\_files\_$i

done

d. График зависимости среднего значения времени выполнения от N

2. Второй эксперимент

a. Скрипт, который, получив параметр N, запускает параллельно N вычислений для разных значений входных параметров (файлов)

#!/bin/bash

n=$1

num=$2

for ((i=1; i<=$n; i++)); do

    ./algorithm2.sh $num/file\_$i.txt &

done

wait

b. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

c. График зависимости среднего значения времени выполнения от N

3. Третий эксперимент

a. Используем два процессора

b. Используем скрипт, который запускает последовательно друг за другом N вычислений для разных значений входных параметров из пункта 1.b

c. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

d. График зависимости среднего значения времени выполнения от N

4. Четвертый эксперимент

a. Используем скрипт, который запускает параллельно друг за другом N вычислений для разных значений входных параметров из пункта 1.b

b. Используем скрипт для записи времени выполнения в файл из пункта 1.c, на выходе получается 20 серий по 10 значений

c. График зависимости среднего значения времени выполнения от N

Выводы:

1. Вычислительно сложный алгоритм

Одно вычислительное ядро: при параллельном запуске время выполнения больше, чем при последовательном, т. к. одно ядро может выполнять в один момент времени только один процесс, поэтому работает столько же + затраты на переключение между процессами.

Два вычислительных ядра: ситуация улучшается при параллельном запуске, так как процессы разделяются между двумя вычислительными ядрами. При последовательном запуске ситуация не меняется, так как нельзя поделить один однопоточный процесс между двумя ядрами и второе просто будет простаивать.

2. Вычислительный алгоритм, который работает с записью и чтением диска

Ситуация такая же, как и в первом эксперименте, но тут влияет не используемый ресурс одного ядра, а количество доступных потоков ввода-вывода для одного ядра. Соответственно при увеличении ядер при параллельном запуске суммарная скорость чтения-записи на диск выше