**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФГБОУ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ**

**«КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

Кафедра «Безопасность информационных и автоматизированных систем»

**Отчет**

Лабораторная работа №4

По дисциплине

«Аппаратные средства ВТ»

Тема: «Реализация параллельных вычислений с помощью многопоточности»

Выполнили студенты гр. Т-40913 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ А.Ю. Бутенко /

Руководитель, ст. преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ В.А. Стукало /

2016

**Постановка задачи:**

Разработать программу расчета числа π путем использования алгоритма Монте-Карло. Реализовать простейшее параллельное вычисление средствами языка Go.

**Цель**:

Дать представление о построении простых параллельных программ на языке параллельного программирования OpenMP; практическое освоение основных директив языка.

**Теоретическое введение**

**Закон Амдала** — иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей. Джин Амдал сформулировал закон в 1967 году, обнаружив простое по существу, но непреодолимое по содержанию ограничение на рост производительности при распараллеливании вычислений: «В случае, когда задача разделяется на несколько частей, суммарное время её выполнения на параллельной системе не может быть меньше времени выполнения самого длинного фрагмента». Согласно этому закону, ускорение выполнения программы за счёт распараллеливания её инструкций на множестве вычислителей ограничено временем, необходимым для выполнения её последовательных инструкций.

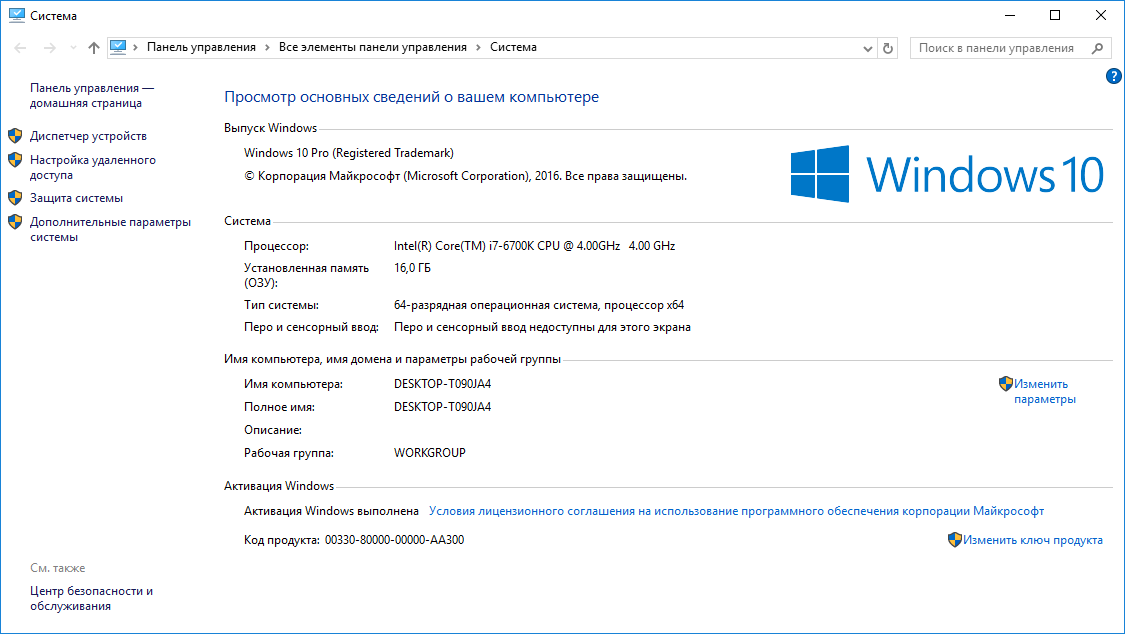
f – доля от общего объема вычислений, которая может быть получена только последовательными расчетами.

(1-f) – доля, которая может быть распараллелена идеально (то есть время вычисления будет обратно пропорциональна числу задействованных ядер).

Так, например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10 кратного ускорения обработки данных.

**Практическая часть**

Данная лабораторная работа проводилась на стационарном ПК с 8-ми ядерным процессором.



**Эксперимент 1:** Отключаем все ядра, кроме одного. Считаем время выполнения программы с одним потоком (три раза), записываем среднее время. Берем два потока, считаем среднее время и т. д до восьми потоков.

На основании этих данных построим график зависимости времени выполнения программы от количества потоков.

**Результат эксперимента 1:**

**Пояснение:** Каждый числовой ряд означает перечень потоков с заданным количеством активированных ядер. Пример: 1 ряд означает, что работает одно ядро и на нем вычисляется число Pi на 1..8 потоках.

**Вывод:** Наблюдаем увеличение производительности только в том случае, когда количество потоков <= количеству ядер. В противном случае, в пределах погрешности, значение не меняется (т.е 8 потоков на одном ядре имеют такую же производительность, что и один поток на той же конфигурации).

**Эксперимент 2:** вычисление максимального ускорения:

**Результат эксперимента 2:**

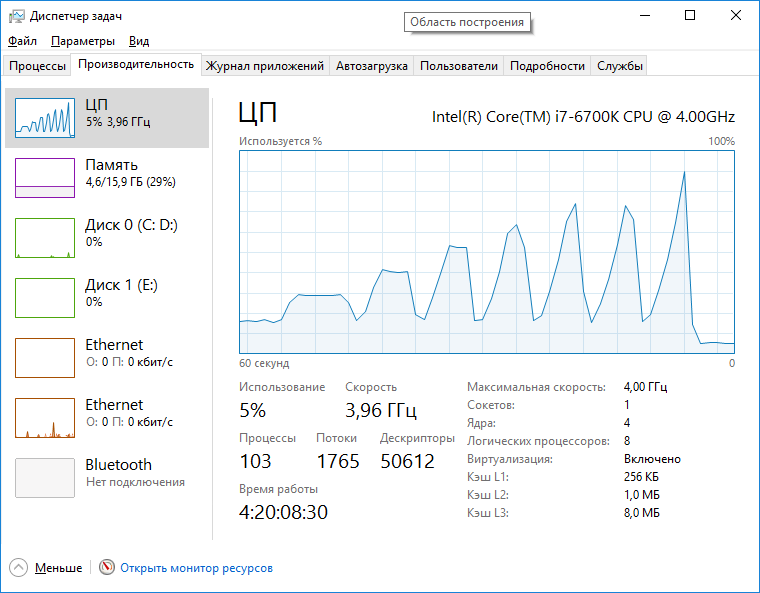
Максимальное ускорение, которое получилось достигнуть, используя 8 ядра S=4,450965.

Из закона Амдала: 4,450965=1/(f+(1-f)/8),

f = 0.1139

**Вывод:** Всего 11.4% кода является последовательным.

**Эксперимент 3:** иллюстрация работы программы с помощью диспетчера задач:



**Эксперимент 4, дополнительный:** работа числового ряда на одном и нескольких ядрах.

Вычисление числа Пи с точностью до 8 знаков с помощью ряда Лейбница.

Программа, работающая на одном ядре, выполнила вычисление за:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 33.9903803s | 34.1125512s | 33.8544853s | 33.8555868s | 33.9382582s |

Программа, работающая на всех доступных ядрах, выполнила вычисление за:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1m11.59751s | 1m10.33565s | 1m11.33561s | 1m11.85344s | 1m11.12552s |

В данном случае, можно сделать вывод, что простые арифметические операции плохо поддаются параллельным вычислениям. Время работы увеличилось, значит расходы на создание и поддержание потоков больше, чем вычислительная сложность алгоритма.

**Вывод:** в процессе использования параллельных вычислений необходимо учитывать то, насколько код, который вы написали, может быть распараллелен. Другими словами, насколько, в итоге, ваш код последовательный.