Правительство Российской Федерации

Федеральное государственное автономное образовательное

учреждение высшего образования "Национальный исследовательский университет "Высшая школа экономики"

Московский институт электроники и математики им. А. Н. Тихонова

Департамент прикладной математики

ОТЧЕТ

по лабораторной работе по дисциплине

«Параллельные вычисления»

Тема работы:

Векторизация и распараллеливание программ

(Вариант 3 для 1.1 и 2 для 1.2)

Выполнил: Студент СКБ171

Лисьев А. Н.

Москва, 2021

# Задача 1.1

Поставленная задача

Вариант: 3 - вещественное сложение, векторное вещественное сложение

- Написать программу, выполняющую многократно (в цикле) заданную арифметическую операцию.

- Замерить время выполнения цикла. По результатам замера получить оценку производительности

микропроцессора на заданной операции (в тактах процессора):

а) используя последовательность зависимых операций ("латентность"),

б) используя последовательность независимых операций ("темп выдачи результатов").

Теория

Для решения данной задачи использовались SIMD команды из списка SSE2:

* **\_mm\_add\_pd(val1, val2)** – возвращает 128-битный вектор, содержащий сумму val1 и val2

Замеры времени производились с помощью библиотеки **std::chrono**. Количество тактов процессора вычислялось с помощью ассемблерной инструкции **rdtsc.**

Описание вычислительной системы

- Имя процессора: Intel(R) Core(TM) i5-9600K CPU @ 3.70GHz

- Скорость процессора: 3.70GHz

- Количество процессоров: 1

- Общее количество ядер: 4

- Память: 16 ГБ

Код программы:

Код представлен в моем github по ссылке:

<https://github.com/alexlis13/parallel1.1>

Результаты программы:

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Результаты:

В ходе проведения практического задания на основании представленных выше результатов можно сделать вывод, что при векторизации сложения время выполнения одного и того же кода сократилось в 2 раза, по сравнению с обычным сложением. Это связано с тем, что регистр, длина которого составляет 128 бит, помещается в 2 значения типа double, что совпадает с теоретическими значениями. Можно добавить, что сократилось и число тактов процессора. Но отношение числа тактов процессов ко времени выполнения не изменяется вследствие того, что производительность процессора не зависит от типа выполняемой операции.

# Задача 1.2

Поставленная задача:

Вариант: 2 – Stencil

Оптимизировать заданную (взятую из открытого доступа) программу автоматически или полуавтоматически с помощью компилятора, а также при желании с помощью intrinsics. Сравнить времена работы следующих вариантов программы:

* Исходная программа, без оптимизации
* Исходная программа, оптимизированная только с помощью ключей компилятора
* Программа, векторизованная полуавтоматически (с помощью директив и ключей компилятора и незначительной правки кода)

Программа, векторизованная и распараллеленная полуавтоматически

Теоретическая часть:

Использованные ключи компилятора:

* ***-O0*** — выключить оптимизацию
* -***Ofast -*** включить максимально возможную высокоуровневую оптимизацию

Использованные директивы:

* ***#pragma omp for simd*** — векторизация цикла (вероятно, это может испортить программу)
* ***#pragma omp parallel for*** — разделить цикл на итерации между потоками

Код программы: <https://github.com/alexlis13/parallel1.1>

Код для интеграции(quick sort): <http://www.codebind.com/cpp-tutorial/cpp-example-quick-sort-algorithm/>

Результаты:

|  |  |  |
| --- | --- | --- |
| **Уровень оптимизации** | **Среднее время(µs)** | **Ускорение** |
| -O0 | 611.2 | нет |
| -O1 | 512.3 | 1.19 |
| -O2 | 278.2 | 2.19 |
| -O3 | 271.7 | 2.24 |
| -Ofast | 270.6 | 2.25 |
| -Ofast + pragma omp for simd | 260.3 | 2.34 |
| -Ofast + pragma omp parallel for | 259.4 | 2.35 |

На графике видно, что начиная с 1 флага происходит ускорение.

|  |  |  |
| --- | --- | --- |
| **Количество потоков** | **Среднее время (µs)** | **Ускорение** |
| 1 | 249 | нет |
| 2 | 248,9 | 1,0004 |
| 3 | 247,6 | 1,0005 |
| 4 | 244,9 | 1.0167 |
| 5 | 242,1 | 1.0285 |
| 6 | 241 | 1.0331 |
| 7 | 240,6 | 1.0349 |
| 8 | 240,5 | 1.0353 |