

Комплексное задание

Выполнил: Селиверстов Михаил

Группа: ИВТ-12М

Вариант 2

Все необходимые программные коды находятся на GitHub:

https://github.com/MichilMIET/complex_task.git

1. Последовательная программа по расчету интеграла

Аналитическое решение:

Для левых прямоугольников:

$$\int_0^1 \frac{4}{1+x^2} dx = h \cdot \sum_{i=0}^{n-1} f(x_i) = \frac{b-a}{n} \cdot \sum_{i=0}^{n-1} f\left(\frac{4}{1+x_i^2}\right)$$

Для правых прямоугольников:

$$\int_0^1 \frac{4}{1+x^2} dx = h \cdot \sum_{i=1}^n f(x_i) = \frac{b-a}{n} \cdot \sum_{i=1}^n f\left(\frac{4}{1+x_i^2}\right)$$

В результате выполнения программы, получаем для каждого из интервалов разбиения значения точности и времени выполнения. Точность я брал в процентах, на мой взгляд это более нагляднее, чем разница в значениях (где 100% полное соответствие с действительным значением). Ниже приведены результаты работы программы для каждого из интервалов.

```
Интервал разбиения: 100
Результат (правого): 3.131575986923128773753433
Точность (правого): 98.9983333333 %
Время выполнения (правого): 0.00002 сек
-----
Результат (левого): 3.1315759869231287737534331
Точность (левого): 98.9983333333 %
Время выполнения (левого): 0.00004 сек
```

Рисунок 1 - Результат работы программы в случае разбиения на 100

```
Интервал разбиения: 1000
Результат (правого): 3.1405924869231229834554142
Точность (правого): 99.8999833333 %
Время выполнения (правого): 0.00012 сек
-----
Результат (левого): 3.1405924869231229834554142
Точность (левого): 99.8999833333 %
Время выполнения (левого): 0.00012 сек
```

Рисунок 2 - Результат работы программы в случае разбиения на 1000

```
Интервал разбиения: 10000
Результат (правого): 3.1414926519232273527393318
Точность (правого): 99.9899998333 %
Время выполнения (правого): 0.00177 сек
-----
Результат (левого): 3.1414926519232273527393318
Точность (левого): 99.9899998333 %
Время выполнения (левого): 0.00170 сек
```

Рисунок 3 - Результат работы программы в случае разбиения на 10000

```
Интервал разбиения: 100000
Результат (правого): 3.1415826535738990976653895
Точность (правого): 99.9989999984 %
Время выполнения (правого): 0.01456 сек
-----
Результат (левого): 3.1415826535738990976653895
Точность (левого): 99.9989999984 %
Время выполнения (левого): 0.01477 сек
```

Рисунок 4 - Результат работы программы в случае разбиения на 100000

```
Интервал разбиения: 1000000
Результат (правого): 3.1415916535910830553746109
Точность (правого): 99.9999000001 %
Время выполнения (правого): 0.14998 сек
-----
Результат (левого): 3.1415916535910830553746109
Точность (левого): 99.9999000001 %
Время выполнения (левого): 0.18178 сек
```

Рисунок 5 - Результат работы программы в случае разбиения на 1000000

Как видно, что с увеличением интервала, увеличивается время выполнения, поскольку алгоритму требуется больше времени на выполнения программы, но возрастает точность, поскольку если интервал сделать бесконечно большим, то число будет бесконечно близко к истинному.

2. Программа по расчету интеграла с использованием нескольких потоков и векторных инструкций

Далее я использовал потоки, а именно `qrag`, автоматическую векторизацию, совместно (`thread`, `mutex`).

Ниже представлены результаты работы программы для каждого интервала разбиения.

```

Интервал разбиения: 100

Без использования потоков:
Точность (правого): 98.9983333333 %
Время выполнения (правого): 0.00003 сек
-----
Точность (левого) = 98.9983333333 %
Время выполнения (левого): 0.00001 сек
*****
С использованием Qpar:
Точность (правого): 98.9983333333 %
Время выполнения (правого): 0.00002 сек
-----
Точность (левого) = 98.9983333333 %
Время выполнения (левого): 0.00002 сек
*****
С использованием автоматической векторизацией:
Точность (правого): 98.9983333333 %
Время выполнения (правого): 0.00002 сек
-----
Точность (левого) = 98.9983333333 %
Время выполнения (левого): 0.00002 сек
*****
С использованием thread и mutex:
Точность (правого) = 99.7999333333 %
Время выполнения (правого): 0.00028 сек
-----
Точность (левого) = 99.7999333333 %
Время выполнения (левого): 0.00416 сек

```

Рисунок 6 - Результат работы программы в случае разбиения на 100

```

Интервал разбиения: 1000

Без использования потоков:
Точность (правого): 99.8999833333 %
Время выполнения (правого): 0.00018 сек
-----
Точность (левого) = 99.8999833333 %
Время выполнения (левого): 0.00016 сек
*****
С использованием Qpar:
Точность (правого): 99.8999833333 %
Время выполнения (правого): 0.00016 сек
-----
Точность (левого) = 99.8999833333 %
Время выполнения (левого): 0.00017 сек
*****
С использованием автоматической векторизацией:
Точность (правого): 99.8999833333 %
Время выполнения (правого): 0.00012 сек
-----
Точность (левого) = 99.8999833333 %
Время выполнения (левого): 0.00012 сек
*****
С использованием thread и mutex:
Точность (правого) = 99.9799933333 %
Время выполнения (правого): 0.00398 сек
-----
Точность (левого) = 99.9799933333 %
Время выполнения (левого): 0.00378 сек

```

Рисунок 7 - Результат работы программы в случае разбиения на 1000

```

Интервал разбиения: 10000

Без использования потоков:
Точность (правого): 99.9899998333 %
Время выполнения (правого): 0.00183 сек
-----
Точность (левого) = 99.9899998333 %
Время выполнения (левого): 0.00183 сек
*****
С использованием Qpar:
Точность (правого): 99.9899998333 %
Время выполнения (правого): 0.00172 сек
-----
Точность (левого) = 99.9899998333 %
Время выполнения (левого): 0.00218 сек
*****
С использованием автоматической векторизацией:
Точность (правого): 99.9899998333 %
Время выполнения (правого): 0.00208 сек
-----
Точность (левого) = 99.9899998333 %
Время выполнения (левого): 0.00185 сек
*****
С использованием thread и mutex:
Точность (правого) = 99.9979999933 %
Время выполнения (правого): 0.01319 сек
-----
Точность (левого) = 99.9979999933 %
Время выполнения (левого): 0.01050 сек

```

Рисунок 8 - Результат работы программы в случае разбиения на 10000

```

Интервал разбиения: 100000

Без использования потоков:
Точность (правого): 99.9989999984 %
Время выполнения (правого): 0.01194 сек
-----
Точность (левого) = 99.9989999984 %
Время выполнения (левого): 0.01224 сек
*****
С использованием Qpar:
Точность (правого): 99.9989999984 %
Время выполнения (правого): 0.01228 сек
-----
Точность (левого) = 99.9989999984 %
Время выполнения (левого): 0.01231 сек
*****
С использованием автоматической векторизацией:
Точность (правого): 99.9989999984 %
Время выполнения (правого): 0.01233 сек
-----
Точность (левого) = 99.9989999984 %
Время выполнения (левого): 0.01224 сек
*****
С использованием thread и mutex:
Точность (правого) = 99.9998000002 %
Время выполнения (правого): 0.10332 сек
-----
Точность (левого) = 99.9998000002 %
Время выполнения (левого): 0.08599 сек

```

Рисунок 9 - Результат работы программы в случае разбиения на 100000

```

Интервал разбиения: 1000000

Без использования потоков:
Точность (правого): 99.9999000001 %
Время выполнения (правого): 0.15757 сек
-----
Точность (левого) = 99.9999000001 %
Время выполнения (левого): 0.16201 сек
*****
С использованием Qrag:
Точность (правого): 99.9999000001 %
Время выполнения (правого): 0.15060 сек
-----
Точность (левого) = 99.9999000001 %
Время выполнения (левого): 0.15589 сек
*****
С использованием автоматической векторизацией:
Точность (правого): 99.9999000001 %
Время выполнения (правого): 0.16230 сек
-----
Точность (левого) = 99.9999000001 %
Время выполнения (левого): 0.15583 сек
*****
С использованием thread и mutex:
Точность (правого) = 99.9999799994 %
Время выполнения (правого): 1.26554 сек
-----
Точность (левого) = 99.9999799994 %
Время выполнения (левого): 1.12959 сек

```

Рисунок 10 - Результат работы программы в случае разбиения на 1000000

Как видно, распараллеливание дает выигрыш во времени, но с увеличением интервала выигрыш становится не столь сильно заметен.

3. Программа с использованием дополнений Intel Cilk Plus языка C++

Далее я добавил цикл `cilk_for`, для распараллеливания вычислений в программе и объединил интервалы на одном скриншоте.

```

Интервал разбиения: 100
Результат (правого): 3.13959198692313 Время выполнения: 0.020171271 сек
Результат (левого): 3.13959198692313 Время выполнения: 0.001588887 сек
*****

Интервал разбиения: 1000
Результат (правого): 3.14139264692314 Время выполнения: 0.000395515 сек
Результат (левого): 3.14139264692314 Время выполнения: 0.000235517 сек
*****

Интервал разбиения: 10000
Результат (правого): 3.14157265352296 Время выполнения: 0.002607328 сек
Результат (левого): 3.14157265352296 Время выполнения: 0.002637194 сек
*****

Интервал разбиения: 100000
Результат (правого): 3.14159065359186 Время выполнения: 0.018837527 сек
Результат (левого): 3.14159065359186 Время выполнения: 0.017865592 сек
*****

Интервал разбиения: 1000000
Результат (правого): 3.14159245358412 Время выполнения: 0.184200815 сек
Результат (левого): 3.14159245358412 Время выполнения: 0.203764092 сек
*****

```

Рисунок 11 - Результат работы программы для всех интервалов разбиения

Как видно время затрачено значительно меньше, чем в предыдущих решениях, но все равно время увеличивается при увеличении интервала разбиения.

Далее использовал Intel Inspector для обнаружения ошибок распараллеливания. Ошибок не обнаружено.

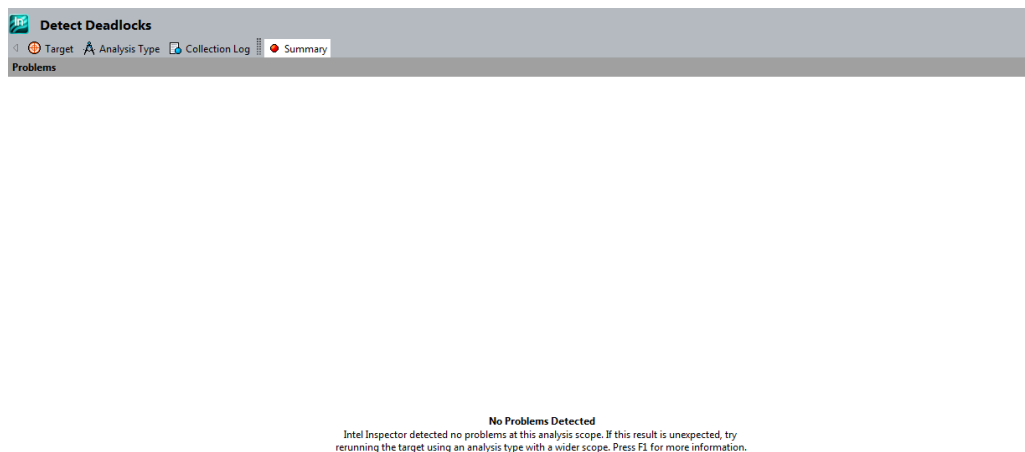


Рисунок 12 - Результат работы Inspector в режиме обнаружения ошибок в потоках

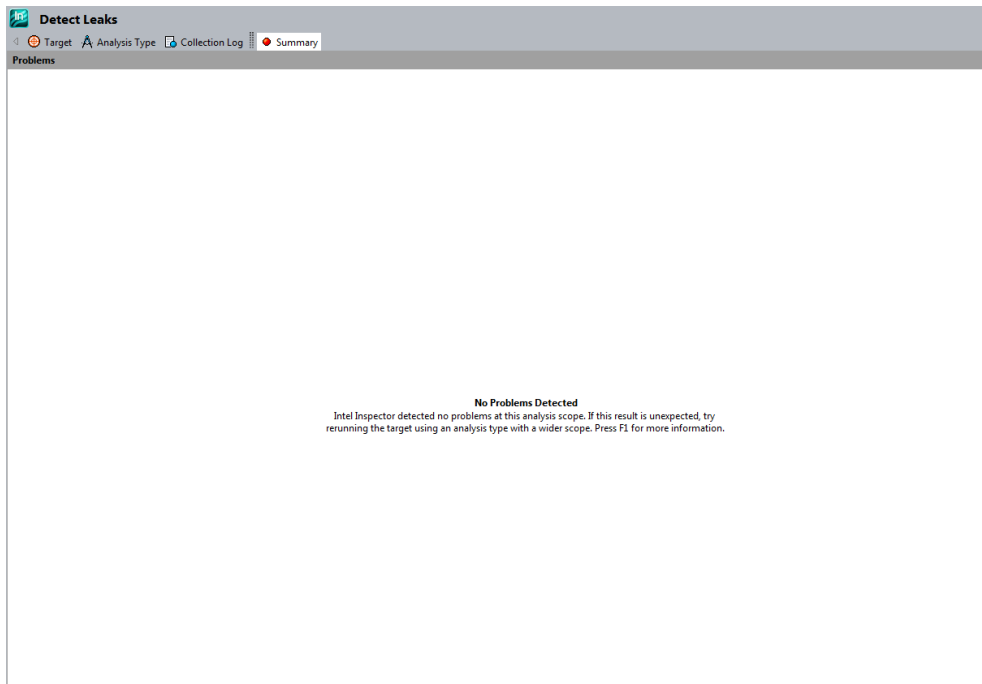


Рисунок 13 - Результат работы Inspector в режиме обнаружения ошибок в памяти

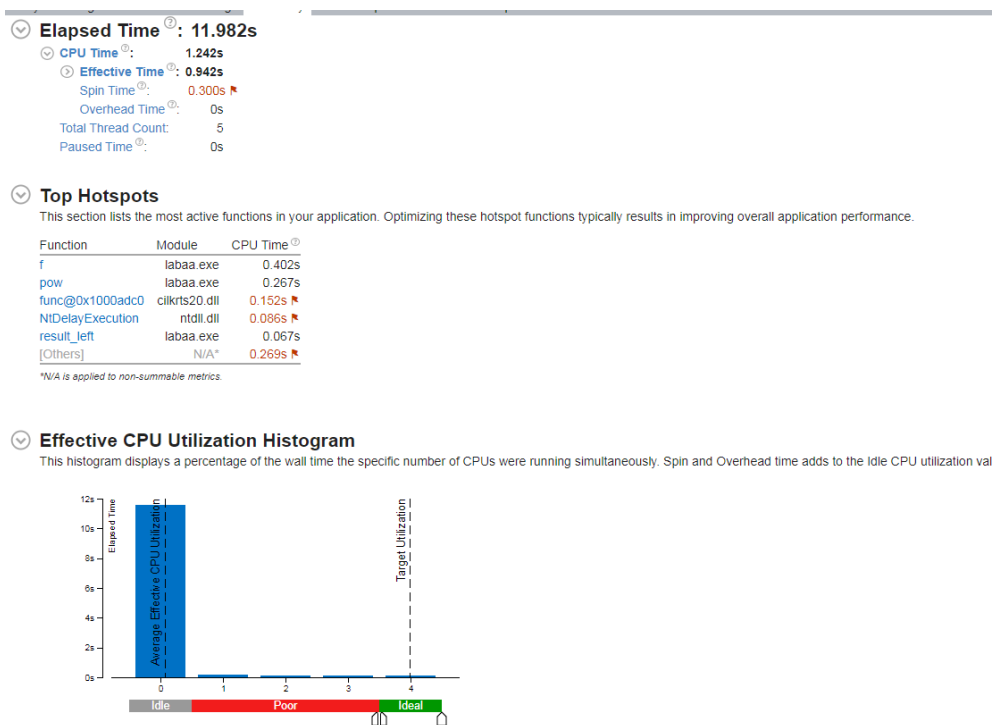


Рисунок 14 - Результат работы VTune, оценка затраченного времени

Вывод: в результате проделанной работы я выяснил как влияют на скорость работы программы параллельные функции. Использование библиотеки Intel cilk оказалось самым оптимальным в плане скорости работы методом.