

Отчёт по лабораторной работе №1

Петрова О. А. группы ИВТ-12М

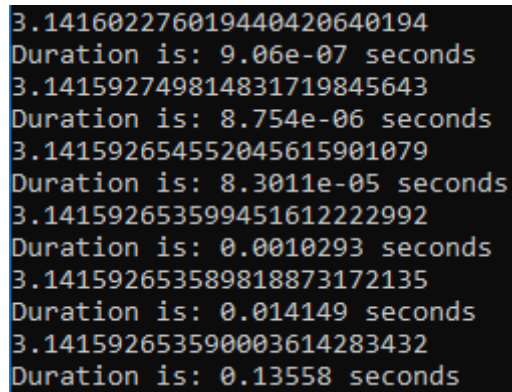
Вариант №12

Интеграл: $\int_{\frac{1}{2}}^1 \frac{6}{\sqrt{x*(2-x)}} dx$

Способ решения: Метод центральных прямоугольников

1. Последовательная программа по расчету интеграла

Результат разработанной последовательной программы изображён на рисунке 1.



```
3.141602276019440420640194
Duration is: 9.06e-07 seconds
3.141592749814831719845643
Duration is: 8.754e-06 seconds
3.141592654552045615901079
Duration is: 8.3011e-05 seconds
3.141592653599451612222992
Duration is: 0.0010293 seconds
3.141592653589818873172135
Duration is: 0.014149 seconds
3.141592653590003614283432
Duration is: 0.13558 seconds
```

Рисунок 1. Результат последовательных вычислений интеграла

Исходя из результатов вычислений можно сделать вывод что с увеличением шага точность повысилась.

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

Результат разработанной программы с WinAPI потоками изображён на рисунке 2, и с стандартными потоками изображён на рисунке 3.

```

=== WinAPI Parallelization ===
No threads:
9.622429647304642230665195e-06
Duration is: 1.207e-06 seconds
Qpar threads:
9.622429647304642230665195e-06
Duration is: 9.05e-07 seconds
No vector:
9.622429647304642230665195e-06
Duration is: 9.06e-07 seconds
=====
No threads:
9.622503860384767904179171e-08
Duration is: 8.452e-06 seconds
Qpar threads:
9.622503860384767904179171e-08
Duration is: 8.452e-06 seconds
No vector:
9.622503860384767904179171e-08
Duration is: 8.452e-06 seconds
=====
No threads:
9.6225249990311567671597e-10
Duration is: 0.00013312 seconds
Qpar threads:
9.6225249990311567671597e-10
Duration is: 8.2709e-05 seconds
No vector:
9.6225249990311567671597e-10
Duration is: 0.00042593 seconds
=====
No threads:
9.658496225029011839069426e-12
Duration is: 0.0012862 seconds
Qpar threads:
9.658496225029011839069426e-12
Duration is: 0.00084219 seconds
No vector:
9.658496225029011839069426e-12
Duration is: 0.0016204 seconds
=====
No threads:
2.575717417130363173782825e-14
Duration is: 0.020096 seconds
Qpar threads:
2.575717417130363173782825e-14
Duration is: 0.0099457 seconds
No vector:
2.575717417130363173782825e-14
Duration is: 0.0095264 seconds
=====
No threads:
2.104982854689296800643206e-13
Duration is: 0.11497 seconds
Qpar threads:
2.104982854689296800643206e-13
Duration is: 0.12729 seconds
No vector:
2.104982854689296800643206e-13
Duration is: 0.098302 seconds
=====

```

Рисунок 2. Результат вычислений с WinAPI потоками

```

=== Normal Parallelization ===
[100] Без потоков: 9.622429647304642230665195e-06
Duration is: 9.06e-07 seconds
С потоками: 3.849e-07
Duration is:5.132e-06 seconds
[1000] Без потоков: 9.622503860384767904179171e-08
Duration is: 8.452e-06 seconds
С потоками: 3.849e-09
Duration is:4.558e-05 seconds
[10000] Без потоков: 9.6225249990311567671597e-10
Duration is: 8.3917e-05 seconds
С потоками: 3.8491e-11
Duration is:0.0039981 seconds
[100000] Без потоков: 9.658496225029011839069426e-12
Duration is: 0.0013874 seconds
С потоками: 3.7259e-13
Duration is:0.021047 seconds
[1000000] Без потоков: 2.575717417130363173782825e-14
Duration is: 0.01154 seconds
С потоками: 3.1974e-14
Duration is:0.12815 seconds
[10000000] Без потоков: 2.104982854689296800643206e-13
Duration is: 0.09841 seconds
С потоками: -1.4744e-13
Duration is:1.1207 seconds

```

Рисунок 3. Результат вычислений со стандартными потоками.

Можно заметить, что с увеличением шага WinAPI потоки продолжают вычислять быстрее линейного способа, а вот стандартные потоки с самого начала вычисляют дольше линейного способа.

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

Для лучшего распараллеливания была использована библиотека Intel Cilk Plus. Стандартный цикл `for` был заменен на `cilk_for`, что позволило распараллелить вычисления. Вычисления изображены на рисунке 4.

```
Let's start!
result = 3.141593e+00 time = 2.771380e-03 step = 100
result = 3.141593e+00 time = 3.199700e-05 step = 1000
result = 3.141593e+00 time = 2.137170e-04 step = 10000
result = 3.141593e+00 time = 1.344486e-03 step = 100000
result = 3.141593e+00 time = 1.487598e-02 step = 1000000
result = 3.141593e+00 time = 1.942514e-01 step = 10000000
```

Рисунок 4. Результат вычислений с Intel Cilk

В результате вычислений видно, что с увеличением шага увеличивается длительность вычислений. К сожалению сам результат был выведен в укороченном виде и не видно всей точности вычислений. Так же можно заметить, что скорость вычислений с помощью `cilk_for` значительно быстрее стандартных и WinAPI потоков.

Для обнаружения ошибки распараллеливания был использован инструмент Intel Inspector. Однако никаких тупиков и гонок данных обнаружено не было. Результат использования инструмента представлен на рисунке 5.



No Problems Detected
Intel Inspector detected no problems at this analysis scope. If this result is unexpected, try rerunning the target using an analysis type with a wider scope. Press F1 for more information.

Рисунок 5. Результат работы Intel Inspector

Так же был проведён анализ программы на эффективность распараллеливания с помощью Intel VTune. Результаты изображена на рисунках 6 и 7.

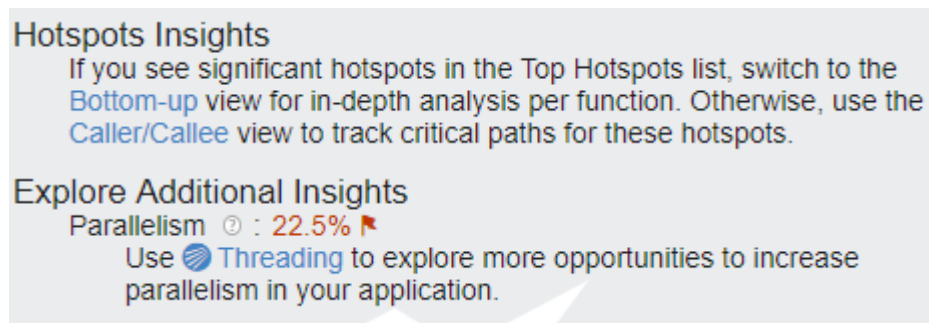


Рисунок 6. Оценка параллелизма

Elapsed Time[?]: 2.434s

CPU Time[?]: 2.439s

Effective Time[?]: 2.193s

Spin Time[?]: 0.245s

Overhead Time[?]: 0s

Total Thread Count: 4

Paused Time[?]: 0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these performance.

Function	Module	CPU Time [?]
f	lab1_4.exe	1.002s
func@0x1002fab4	libmmd.dll	0.545s
compute	lab1_4.exe	0.352s
NtDelayExecution	ntdll.dll	0.238s
sqrt	libmmd.dll	0.153s
[Others]		0.148s

**N/A is applied to non-summable metrics.*

Effective CPU Utilization Histogram

This histogram displays a percentage of the wall time the specific number of C

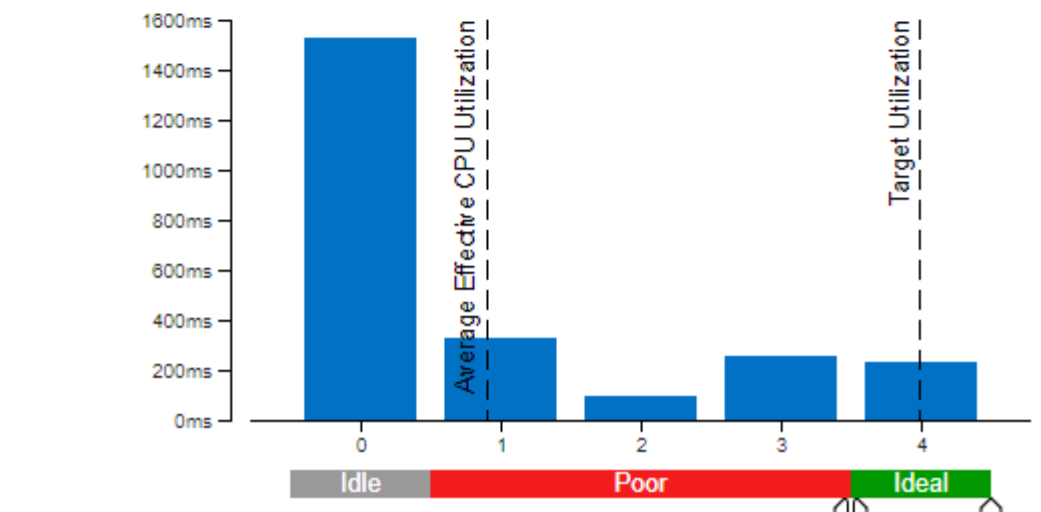
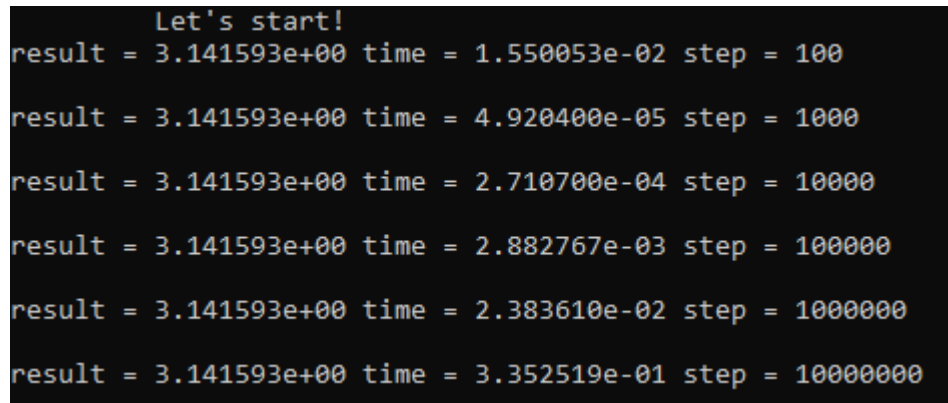


Рисунок 7. Результат работы VTune по затраченному времени и используемым CPU

4. Программа с использованием шаблонов TBB

Из данной библиотеки использовалась шаблонная функция `parallel_for`, на вход которой подаётся коллекция (начало и конец) и функтор.

Результаты вычислений изображён на рисунке 8.



```
Let's start!
result = 3.141593e+00 time = 1.550053e-02 step = 100
result = 3.141593e+00 time = 4.920400e-05 step = 1000
result = 3.141593e+00 time = 2.710700e-04 step = 10000
result = 3.141593e+00 time = 2.882767e-03 step = 100000
result = 3.141593e+00 time = 2.383610e-02 step = 1000000
result = 3.141593e+00 time = 3.352519e-01 step = 10000000
```

Рисунок 8. Результат вычислений с использованием библиотеки шаблонов Intel TBB

Вывод: Исходя из полученных результатов можно сказать что они не сильно отличаются, и можно заметить, что наилучшим по скорости вычисления оказался Intel Cilk, но Intel TBB довольно близок по скорости. Так или иначе программы с распараллеливанием всё равно оказались быстрее линейного вычисления.