**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра Вычислительной техники**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Параллельные алгоритмы и системы»**

**Тема: Методы оптимизации кода на примере вычисления числа пи**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9307 |  | Немцов М.Л. |
| Преподаватель |  | Пазников А.А. |

Санкт-Петербург

2023

# ВВЕДЕНИЕ

Цель: Изучить основные методы оптимизации кода.

Задача: реализовать алгоритм подсчёта числа пи, после оптимизировать код.

**Подготовка**

Характеристика ВМ:

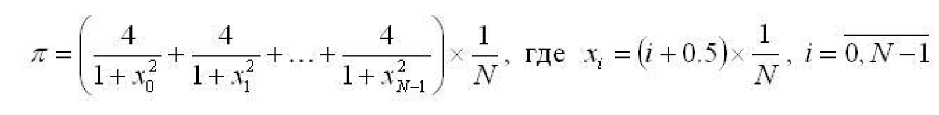
CPU: i7-7700F, 4 ядра;

RAM: DDR4 16GB 3200;

OC: GNU/Linux Ubuntu;   
Компиляторы: gcc (12.2.1), clang (15.0.7).

# 1. Изначальный вариант подсчёта числа пи (без оптимизаций)

Расчет числа пи осуществлялся по следующей формуле:



Реализация :

float calc\_pi(unsigned N\_iters)  
{  
  
 float pi = 0.0;  
 float x\_i = 0.0;  
  
 for(unsigned i = 0; i < N\_iters; ++i)  
 {  
 x\_i = (i + 0.5) / N\_iters;  
 x\_i = 4.0 / (1 + x\_i\*x\_i);  
 pi += x\_i;  
 }  
 pi /= N\_iters;  
  
 return pi;  
}

Компиляция:

> CC lab1\_1.c -o lab1\_1 -O0 -Wall

Исходный код программы lab1\_1.c представлен в приложении.

Время выполнения.

> ./lab1\_1  
# CLANG:  
# CPU time spent: 188.077862 sec (188077862 us)  
# Real time spent: 190.108846 sec (190108846 us)  
  
# GCC:  
# CPU time spent: 31.764762 sec (31764762 us)  
# Real time spent: 31.878273 sec (31878273 us)

Стоит заметить, что компилятор gcc сделал некую оптимизацию, хотя использовался флаг -O0 при компиляции.

Смотрим, что говорит профайлер:

> perf record ./lab1\_1  
> perf report  
# Samples: 767K of event 'cycles:u', Event count (approx.): 731253387448  
# Overhead Command Shared Object Symbol  
# 99,97% lab1\_1 lab1\_1 [.] calc\_pi  
# 0,03% lab1\_1 [unknown] [k] 0xffffffff840018f7  
# 0,00% lab1\_1 ld-linux-x86-64.so.2 [.] 0x000000000000ebf4  
# 0,00% lab1\_1 ld-linux-x86-64.so.2 [.] 0x0000000000015060

Видно, что оптимизировать необходимо функцию calc\_pi.:

for(unsigned i = 0; i < N\_iters; ++i){  
 x\_i = (i + 0.5) / N\_iters;  
 x\_i = 4.0 / (1 + x\_i\*x\_i);  
 pi += x\_i;  
}

Метрики:

> perf stat -B -e task-clock,context-switches,cpu-migrations,cycles,instructions,cache-references,cache-misses,branches,branch-misses,migrations,page-faults ./lab1\_1  
# Performance counter stats for './lab1\_1': CLANG  
  
# 188 683,73 msec task-clock:u # 0,992 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 730 494 788 880 cycles:u # 3,872 GHz   
# 290 029 452 924 instructions:u # 0,40 insn per cycle   
# 2 300 472 cache-references:u # 12,192 K/sec   
# 1 001 822 cache-misses:u # 43,549 % of all cache refs   
# 20 002 310 027 branches:u # 106,010 M/sec   
# 4 875 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 58 page-faults:u # 0,307 /sec   
  
# Performance counter stats for './lab1\_1': GCC  
  
# 31 824,43 msec task-clock:u # 0,998 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 123 402 207 509 cycles:u # 3,878 GHz   
# 330 033 224 996 instructions:u # 2,67 insn per cycle   
# 259 894 cache-references:u # 8,166 K/sec   
# 144 140 cache-misses:u # 55,461 % of all cache refs   
# 50 005 082 642 branches:u # 1,571 G/sec   
# 4 743 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 58 page-faults:u # 1,822 /sec

В качестве метрик будут использоваться:

* время выполнения программы: Время в секундах выполнения.
* instructions: IPC (instructions per second).
* cache-misses: Миссы по кэшу.

Сведём в таблицу:

|  | время | IPC | cache-misses |
| --- | --- | --- | --- |
| gcc | 31.7 | 2,67 | 55,5 |
| clang | 188.1 | 0,40 | 43,5 |

Нужно отметить, что в программе также присутствует “разогрев” и число подсчёта числа пи всегда равняется тысячи.

# 2. Оптимизация кода с помощью data parallelism (intrinsic)

Можно заметить, что считать составляющие числа пи можно и “одновременно”. Хорошим решением будет использование векторных инструкций и векторных регистров.

float calc\_pi(unsigned N\_iters)  
{  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
  
 ALIGNED\_(32) float vres[8];  
 \_\_m256 onem = \_mm256\_set1\_ps(1.0);  
 \_\_m256 Nm = \_mm256\_set1\_ps(N\_f);  
 \_\_m256 buffm;  
  
 for(unsigned i = 0; i < N\_iters; i+=8)  
 {  
 float j = (float)i + 0.5;  
 buffm = \_mm256\_set\_ps(j, j+1.0, j+2.0, j+3.0, j+4.0, j+5.0, j+6.0, j+7.0);  
 buffm = \_mm256\_div\_ps(buffm, Nm);  
 buffm = \_mm256\_mul\_ps(buffm, buffm);  
 buffm = \_mm256\_add\_ps(buffm, onem);  
 buffm = \_mm256\_div\_ps(onem, buffm);  
  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 \_mm256\_store\_ps(vres, buffm);  
 pi += vres[0] + vres[7];  
 }  
  
 pi \*= 4.0;  
 pi /= N\_iters;  
  
 return pi;  
}

Компилируем:

> CC lab1\_2.c -o lab1\_2 -O0 -Wall -mavx2

Исходный код программы lab1\_2.c представлен в приложении.

Смотрим, как изменилась ситуация:

> ./lab1\_2  
# CLANG:  
# CPU time spent: 64.184793 sec (64184793 us)  
# Real time spent: 64.875665 sec (64875665 us)  
  
# GCC:  
# CPU time spent: 56.264375 sec (56264375 us)  
# Real time spent: 56.851376 sec (56851376 us)

Смотрим, что скажет perf.

> perf stat -B -e task-clock,context-switches,cpu-migrations,cycles,instructions,cache-references,cache-misses,branches,branch-misses,migrations,page-faults ./lab1\_2  
# Performance counter stats for './lab1\_2': CLANG  
  
# 64 270,28 msec task-clock:u # 0,998 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 249 108 262 806 cycles:u # 3,876 GHz   
# 148 765 224 069 instructions:u # 0,60 insn per cycle   
# 822 699 cache-references:u # 12,801 K/sec   
# 658 200 cache-misses:u # 60,042 % of all cache refs   
# 2 500 371 926 branches:u # 38,904 M/sec   
# 4 767 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 57 page-faults:u # 0,887 /sec  
  
# Performance counter stats for './lab1\_2': GCC  
  
# 56 525,70 msec task-clock:u # 0,991 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 218 409 933 244 cycles:u # 3,864 GHz   
# 145 014 813 611 instructions:u # 0,66 insn per cycle   
# 1 410 503 cache-references:u # 24,953 K/sec   
# 751 645 cache-misses:u # 53,289 % of all cache refs   
# 3 750 524 261 branches:u # 66,351 M/sec   
# 5 748 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 61 page-faults:u # 1,079 /sec

Сведём в таблицу:

|  | время | IPC | cache-misses |
| --- | --- | --- | --- |
| gcc | 56.3 | 0,66 | 53,3 |
| clang | 64.2 | 0,60 | 60,0 |

Здесь нужно отметить, что метрика cache-misses сильно менялась от замера к замеру.

# 3. Оптимизация кода с помощью openmp

Перепишем функцию под openmp.

float calc\_pi(unsigned N\_iters){  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
 #pragma omp parallel  
 {  
 unsigned th\_n = omp\_get\_num\_threads();  
 unsigned th\_i = omp\_get\_thread\_num();  
 unsigned iter\_per\_th = N\_iters / th\_n;  
 unsigned lb = iter\_per\_th\*th\_i;  
 unsigned ub = 5051;  
 if(th\_i == th\_n-1)  
 ub = N\_iters-1;  
 else  
 ub = lb + iter\_per\_th-1;  
 float pi\_local = 0.0;  
 ALIGNED\_(32) float vres[8];  
 \_\_m256 onem = \_mm256\_set1\_ps(1.0);  
 \_\_m256 Nm = \_mm256\_set1\_ps(N\_f);  
 \_\_m256 buffm;  
 for(unsigned i = lb; i <= ub; i+=8)  
 {  
 float j = (float)i + 0.5;  
 buffm = \_mm256\_set\_ps(j, j+1.0, j+2.0, j+3.0, j+4.0, j+5.0, j+6.0, j+7.0);  
 buffm = \_mm256\_div\_ps(buffm, Nm);  
 buffm = \_mm256\_mul\_ps(buffm, buffm);  
 buffm = \_mm256\_add\_ps(buffm, onem);  
 buffm = \_mm256\_div\_ps(onem, buffm);  
  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 \_mm256\_store\_ps(vres, buffm);  
 pi\_local += vres[0] + vres[7];  
 }  
 #pragma omp atomic  
 pi += pi\_local;  
 }  
 pi \*= 4.0;  
 pi /= N\_iters;  
  
 return pi;  
}

Компилируем:

> CC lab1\_3.c -o lab1\_3 -O0 -Wall -mavx2 -fopenmp

Исходный код программы lab1\_3.c представлен в приложении.

Сделаем замеры:

> ./lab1\_3  
# CLANG:  
# CPU time spent: 64.541564 sec (64541564 us)  
# Real time spent: 10.787499 sec (10787499 us)  
  
# GCC:  
# CPU time spent: 56.361458 sec (56361458 us)  
# Real time spent: 9.429584 sec (9429584 us)

При замерах не были запущены какие-либо другие приложения и программы.

> perf stat -B -e task-clock,context-switches,cpu-migrations,cycles,instructions,cache-references,cache-misses,branches,branch-misses,migrations,page-faults ./lab1\_3  
# Performance counter stats for './lab1\_3': CLANG  
  
# 64 717,34 msec task-clock:u # 5,991 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 251 040 117 627 cycles:u # 3,879 GHz   
# 149 517 410 100 instructions:u # 0,60 insn per cycle   
# 2 372 767 cache-references:u # 36,664 K/sec   
# 64 480 cache-misses:u # 2,718 % of all cache refs   
# 2 742 245 558 branches:u # 42,373 M/sec   
# 184 774 branch-misses:u # 0,01% of all branches   
# 0 migrations:u # 0,000 /sec   
# 228 page-faults:u # 3,523 /sec   
  
# Performance counter stats for './lab1\_3': GCC  
  
# 57 339,27 msec task-clock:u # 5,955 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 222 462 812 833 cycles:u # 3,880 GHz   
# 146 487 026 305 instructions:u # 0,66 insn per cycle   
# 1 515 921 cache-references:u # 26,438 K/sec   
# 45 507 cache-misses:u # 3,002 % of all cache refs   
# 5 063 664 678 branches:u # 88,311 M/sec   
# 53 451 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 85 page-faults:u # 1,482 /sec

Сведём в таблицу:

|  | время | IPC | cache-misses |
| --- | --- | --- | --- |
| gcc | 9.4 | 0,66 | 3 |
| clang | 10.8 | 0,60 | 2,7 |

Сделаем таблицу замера времени в зависимости от кол-ва потоков. На i7-7700 4 физических потока и 4 логических потока.

| потоки | 1 | 2 | 4 | 6 | 8 | 12 |
| --- | --- | --- | --- | --- | --- | --- |
| время | 54.6 | 27.3 | 14.1 | 9.4 | 14.0 | 11 |

# 4. Оптимизация кода с помощью ассемблерной вставки

Перепишем функцию calc\_pi:

float calc\_pi(unsigned N\_iters)  
{  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
  
 #pragma omp parallel  
 {  
 unsigned th\_n = omp\_get\_num\_threads();  
 unsigned th\_i = omp\_get\_thread\_num();  
 unsigned iter\_per\_th = N\_iters / th\_n;  
 unsigned lb = iter\_per\_th\*th\_i;  
 unsigned ub = 5051;  
 if(th\_i == th\_n-1)  
 ub = N\_iters-1;  
 else  
 ub = lb + iter\_per\_th-1;  
  
 float pi\_local = 0.0;  
  
 ALIGNED\_(32) float vres[8];  
 ALIGNED\_(32) float v8[8]; ALIGNED\_(32) float onev[8]; ALIGNED\_(32) float Nm[8];  
 onev[0] = 1.0; onev[1] = 1.0; onev[2] = 1.0; onev[3] = 1.0;  
 onev[4] = 1.0; onev[5] = 1.0; onev[6] = 1.0; onev[7] = 1.0;  
 Nm[0] = N\_f; Nm[1] = N\_f; Nm[2] = N\_f; Nm[3] = N\_f;  
 Nm[4] = N\_f; Nm[5] = N\_f; Nm[6] = N\_f; Nm[7] = N\_f;  
  
 for(unsigned i = lb; i <= ub; i+=8)  
 {  
 float j = (float)i + 0.5;  
 v8[0] = j ; v8[1] = j+1.0; v8[2] = j+2.0; v8[3] = j+3.0;  
 v8[4] = j + 4.0; v8[5] = j+5.0; v8[6] = j+6.0; v8[7] = j+7.0;  
 \_\_asm\_\_ \_\_volatile\_\_(  
 "vmovaps (%[v8]), %%ymm0 \n\t"  
 "vmovaps (%[Nm]), %%ymm1 \n\t"  
 "vmovaps (%[onev]), %%ymm2 \n\t"  
 "vdivps %%ymm1, %%ymm0, %%ymm0 \n\t"  
 "vmulps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vaddps %%ymm0, %%ymm2, %%ymm0 \n\t"  
 "vdivps %%ymm0, %%ymm2, %%ymm0 \n\t"  
 "vhaddps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vhaddps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vmovaps %%ymm0, %[res] \n\t"  
 : [res] "=m"(\*vres)  
 : [v8] "r"(v8), [Nm] "r"(Nm), [onev] "r"(onev)  
 : "%ymm0", "%ymm1", "%ymm2"  
 );  
 pi\_local += vres[0] + vres[7];  
 }  
  
 #pragma omp atomic  
 pi += pi\_local;  
 }  
  
 pi \*= 4.0;  
 pi /= N\_iters;  
  
 return pi;  
}

Компилируем:

> CC lab1\_4.c -o lab1\_4 -O0 -Wall -fopenmp

Исходный код программы lab1\_4.c представлен в приложении.

Сделаем замеры:

> ./lab1\_4  
# CLANG:  
# CPU time spent: 70.223842 sec (70223842 us)  
# Real time spent: 11.750158 sec (11750158 us)  
  
# GCC:  
# CPU time spent: 41.517682 sec (41517682 us)  
# Real time spent: 6.941952 sec (6941952 us)

Вот что perf говорит:

> perf stat -B -e task-clock,context-switches,cpu-migrations,cycles,instructions,cache-references,cache-misses,branches,branch-misses,migrations,page-faults ./lab1\_4  
# Performance counter stats for './lab1\_4': CLANG  
  
# 70 366,40 msec task-clock:u # 5,967 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 272 702 829 419 cycles:u # 3,875 GHz   
# 95 982 680 986 instructions:u # 0,41 insn per cycle   
# 5 318 969 cache-references:u # 75,590 K/sec   
# 100 485 cache-misses:u # 1,889 % of all cache refs   
# 2 814 020 274 branches:u # 39,991 M/sec   
# 223 127 branch-misses:u # 0,01% of all branches   
# 0 migrations:u # 0,000 /sec   
# 228 page-faults:u # 3,240 /sec  
  
# Performance counter stats for './lab1\_4': GCC  
  
# 41 703,38 msec task-clock:u # 5,967 CPUs utilized   
# 0 context-switches:u # 0,000 /sec   
# 0 cpu-migrations:u # 0,000 /sec   
# 161 701 029 546 cycles:u # 3,877 GHz   
# 78 869 553 297 instructions:u # 0,49 insn per cycle   
# 1 880 643 cache-references:u # 45,096 K/sec   
# 29 077 cache-misses:u # 1,546 % of all cache refs   
# 5 032 042 739 branches:u # 120,663 M/sec   
# 62 215 branch-misses:u # 0,00% of all branches   
# 0 migrations:u # 0,000 /sec   
# 83 page-faults:u # 1,990 /sec

Сведём в таблицу:

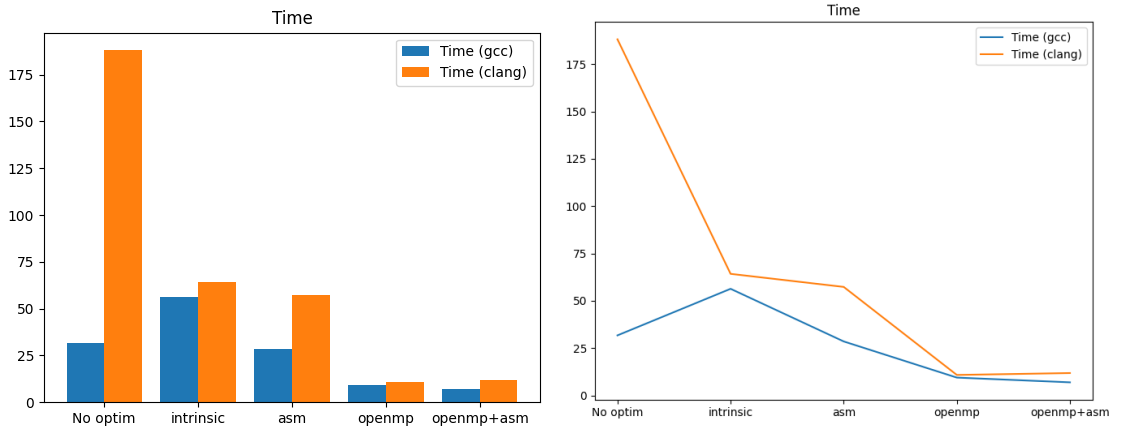
|  | время | IPC | cache-misses |
| --- | --- | --- | --- |
| gcc | 6.9 | 0,49 | 1,5 |
| clang | 11.8 | 0,41 | 1,9 |

# Графики

Сделаем общую таблицу.

|  | время, с | IPC | cache-misses, % | Компилятор |
| --- | --- | --- | --- | --- |
| Без оптимизаций | 31.7 | 2,67 | 55,5 | gcc |
| intrinsic | 56.3 | 0,66 | 53,3 | gcc |
| inline assembly | 28.5 | 0,50 | 30,5 | gcc |
| openmp | 9.4 | 0,66 | 3 | gcc |
| openmp+inline assembly | 6.9 | 0,49 | 1,5 | gcc |
| Без оптимизаций | 188.1 | 0,40 | 43,5 | clang |
| intrinsic | 64.2 | 0,60 | 60,0 | clang |
| inline assembly | 57.3 | 0,35 | 32,7 | clang |
| openmp | 10.8 | 0,60 | 2,7 | clang |
| openmp+inline assembly | 11.8 | 0,41 | 1,9 | clang |

График зависимости времени выполнения от этапа оптимизации:



По графику видно, что в целом оптимизация кода, осущевстлённая в данной лабораторной работе не была напрасной, она дала прирост производительности.

График зависимости ускорения от этапа оптимизации:

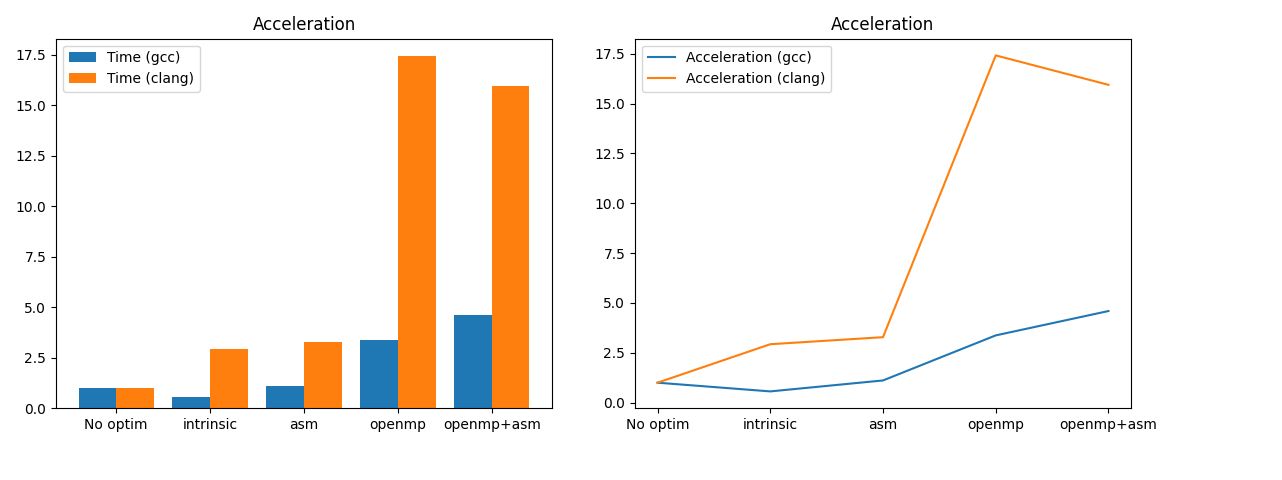


График зависимости IPC от этапа оптимизации:

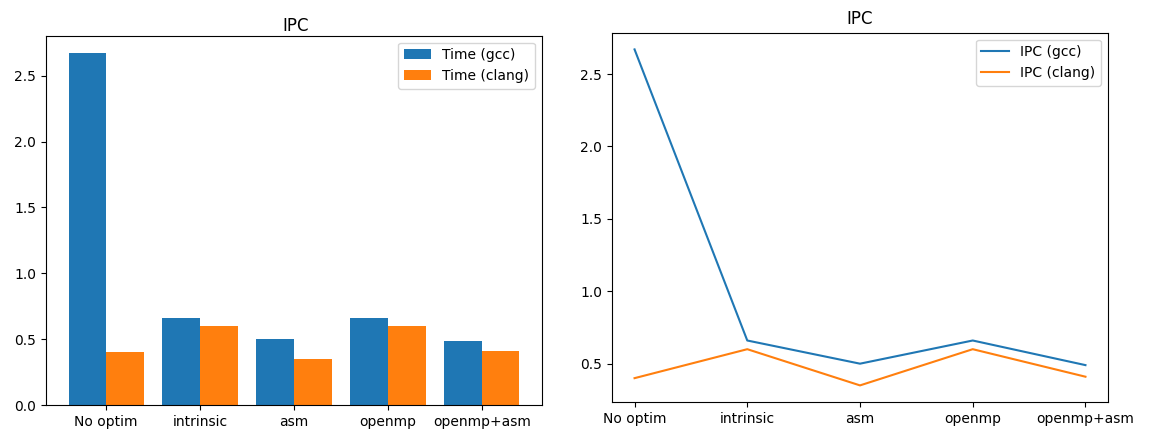


График зависимости cache-misses от этапа оптимизации:

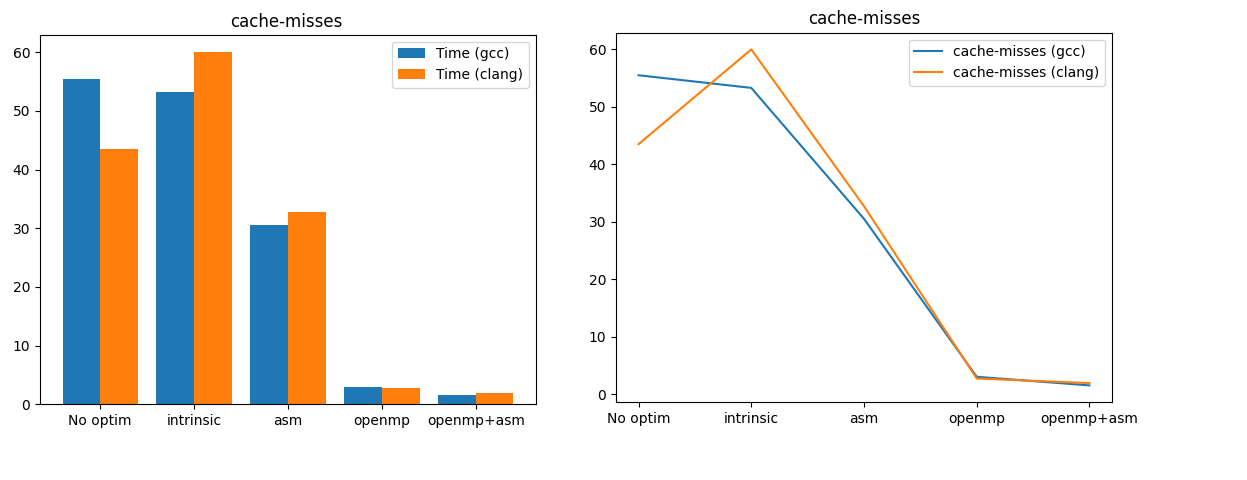


График зависимости времени выполнения от количества потоков:

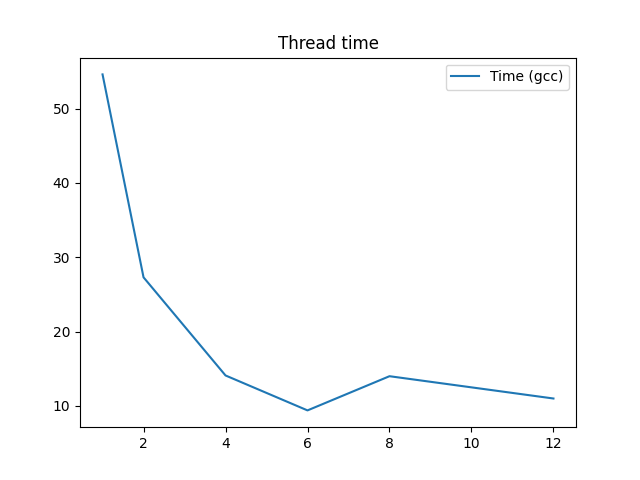
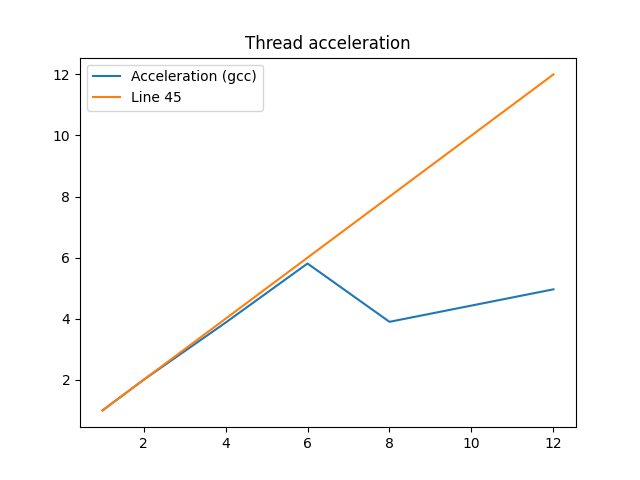


График зависимости ускорения от количества потоков:



# Выводы

Анализируя график зависимости времени выполнения и ускорения можно сделать несколько выводов. Скомпилированный код зависит от компилятора, хоть и была отключена оптимизация во всех случаях компилирования, gcc что-то всё равно оптимизировал.

Рассматривая график IPC от этапа оптимизации, можно сказать, что количество инструкций за такт при использовании ассемблерной вставки уменьшилось. Такое могло произойти из-за того, что перед тем, как выполнить векторную команду раньше был большой блок обращения к памяти, а уже потом выполнялась сама векторная инструкция. Таки образом можно было конвейеризировать. После ассемблерной вставки ситуация такая: сначала много комманд по обращению к памяти и инициализация векторных регистров, а потом уже сразу много подряд векторных комманд. Такое конвейеризировать сложно/невозможно. А так как этот участок кода как раз и выполняется больше всего на процессоре, это дало весомый вклад в получившийся график.

По графику кэш-миссов от этапа оптимизации нужно сделать несколько замечаний. Во-первых, от замера к замеру в отличие от остальных метрик кэш-миссы сильно разнились (в результатах вредставлены средние значения). Во-вторых, можно увидеть очень сильный прирост в кэш-попаданиях, когда речь идёт об openmp. Связано это с тем, что раньше один поток “брал” на себя один большой блок данных, и поэтому часто приходилось в кэш загружать следующую порцию. Ведь сами кэш-мисы считается, как cache-references / cache-misses. Когда много потоков, то каждый поток “берёт” на себя уже блок для вычислений сильно меньший. Меньше раз надо свапать данные в кэше, а отсюда и такие цифры, так как cache-references общий, а cache-misses для каждого потока свой.

Смотря на график зависимости ускорения от кол-ва потоков, можно сказать, что зависимость почти идеальная/линейная. То, что ускорение начинает падать около 6 потоков связано с тем, что в вычислительной системе горлышко от бутылки - это оперативная память. Слишком медленная для процессора, особенно, когда оттуда/туда берут/загружают данные сразу несколько ядер процессора.

# ПРИЛОЖЕНИЕ

Листинг lab1\_1.c:

/\*No optimization\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <sys/time.h>  
  
#define N\_do 1000  
#define N 10000000 // KPATHO 8  
#define N\_do\_hotter 100  
#define N\_hotter 10000 // KPATHO 8  
  
float calc\_pi(unsigned N\_iters);  
  
int main()  
{  
 clock\_t begin\_cpu, end\_cpu;  
 struct timeval begin\_time, end\_time;  
  
 /\*PA3OrPEB\*/  
 for(unsigned i = 0; i < N\_do\_hotter; ++i)  
 calc\_pi(N\_hotter);  
  
 gettimeofday(&begin\_time, 0);  
 begin\_cpu = clock();  
  
 for(unsigned i = 0; i < N\_do; ++i)  
 calc\_pi(N);  
  
 end\_cpu = clock();  
 gettimeofday(&end\_time, 0);  
  
 long time\_seconds = end\_time.tv\_sec - begin\_time.tv\_sec;  
 long time\_microseconds = end\_time.tv\_usec - begin\_time.tv\_usec;  
 double time\_elapsed = time\_seconds + time\_microseconds\*1e-6;  
 long time\_elapsed\_mcs = (long)(time\_elapsed\*1000000 + 0.5);  
  
 double cputime\_spent = (double)(end\_cpu - begin\_cpu) / CLOCKS\_PER\_SEC;  
 long cputime\_spent\_mcs = (long)(cputime\_spent\*1000000 + 0.5);  
  
 printf("CPU time spent: %f sec (%ld us)\n", cputime\_spent, cputime\_spent\_mcs);  
 printf("Real time spent: %f sec (%ld us)\n", time\_elapsed, time\_elapsed\_mcs);  
}  
  
float calc\_pi(unsigned N\_iters)  
{  
  
 float pi = 0.0;  
 float x\_i = 0.0;  
  
 for(unsigned i = 0; i < N\_iters; ++i)  
 {  
 x\_i = (i + 0.5) / N\_iters;  
 x\_i = 4.0 / (1 + x\_i\*x\_i);  
 pi += x\_i;  
 }  
 pi /= N\_iters;  
 // printf("%.10lf\n", pi);  
  
 return pi;  
}

Листинг lab1\_2.c:

/\*No optimization\*/  
/\*Data parallelism\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <sys/time.h>  
  
#include <immintrin.h> // AVX  
  
#if defined(\_MSC\_VER)  
#define ALIGNED\_(x) \_\_declspec(align(x))  
#else  
#if defined(\_\_GNUC\_\_)  
#define ALIGNED\_(x) \_\_attribute\_\_ ((aligned(x)))  
#endif  
#endif  
  
  
#define N\_do 1000  
#define N 10000000 // KPATHO 8  
#define N\_do\_hotter 100  
#define N\_hotter 10000 // KPATHO 8  
  
float calc\_pi(unsigned N\_iters);  
  
int main()  
{  
 clock\_t begin\_cpu, end\_cpu;  
 struct timeval begin\_time, end\_time;  
  
 /\*PA3OrPEB\*/  
 for(unsigned i = 0; i < N\_do\_hotter; ++i)  
 calc\_pi(N\_hotter);  
  
 gettimeofday(&begin\_time, 0);  
 begin\_cpu = clock();  
  
 for(unsigned i = 0; i < N\_do; ++i)  
 calc\_pi(N);  
  
 end\_cpu = clock();  
 gettimeofday(&end\_time, 0);  
  
 long time\_seconds = end\_time.tv\_sec - begin\_time.tv\_sec;  
 long time\_microseconds = end\_time.tv\_usec - begin\_time.tv\_usec;  
 double time\_elapsed = time\_seconds + time\_microseconds\*1e-6;  
 long time\_elapsed\_mcs = (long)(time\_elapsed\*1000000 + 0.5);  
  
 double cputime\_spent = (double)(end\_cpu - begin\_cpu) / CLOCKS\_PER\_SEC;  
 long cputime\_spent\_mcs = (long)(cputime\_spent\*1000000 + 0.5);  
  
 printf("CPU time spent: %f sec (%ld us)\n", cputime\_spent, cputime\_spent\_mcs);  
 printf("Real time spent: %f sec (%ld us)\n", time\_elapsed, time\_elapsed\_mcs);  
}  
  
// https://doc.rust-lang.org/beta/core/arch/x86\_64/index.html  
float calc\_pi(unsigned N\_iters)  
{  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
  
 ALIGNED\_(32) float vres[8];  
 \_\_m256 onem = \_mm256\_set1\_ps(1.0);  
 \_\_m256 Nm = \_mm256\_set1\_ps(N\_f);  
 \_\_m256 buffm;  
  
 for(unsigned i = 0; i < N\_iters; i+=8)  
 {  
 float j = (float)i + 0.5;  
 buffm = \_mm256\_set\_ps(j, j+1.0, j+2.0, j+3.0, j+4.0, j+5.0, j+6.0, j+7.0);  
 buffm = \_mm256\_div\_ps(buffm, Nm);  
 buffm = \_mm256\_mul\_ps(buffm, buffm);  
 buffm = \_mm256\_add\_ps(buffm, onem);  
 buffm = \_mm256\_div\_ps(onem, buffm);  
  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 \_mm256\_store\_ps(vres, buffm);  
 pi += vres[0] + vres[7];  
 }  
  
 pi \*= 4.0;  
 pi /= N\_iters;  
 // printf("%.10lf\n", pi);  
  
 return pi;  
}

Листинг lab1\_3.c:

/\*No optimization\*/  
/\*Data parallelism\*/  
/\*Thread parallelism\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <sys/time.h>  
#include <malloc.h>  
  
#include <immintrin.h> // AVX  
  
#include <omp.h>  
  
#if defined(\_MSC\_VER)  
#define ALIGNED\_(x) \_\_declspec(align(x))  
#else  
#if defined(\_\_GNUC\_\_)  
#define ALIGNED\_(x) \_\_attribute\_\_ ((aligned(x)))  
#endif  
#endif  
  
  
#define N\_do 1000  
#define N 10000000 // KPATHO 8  
#define N\_do\_hotter 100  
#define N\_hotter 10000 // KPATHO 8  
  
float calc\_pi(unsigned N\_iters);  
  
int main()  
{  
 clock\_t begin\_cpu, end\_cpu;  
 struct timeval begin\_time, end\_time;  
  
 /\*PA3OrPEB\*/  
 for(unsigned i = 0; i < N\_do\_hotter; ++i)  
 calc\_pi(N\_hotter);  
  
 gettimeofday(&begin\_time, 0);  
 begin\_cpu = clock();  
  
 for(unsigned i = 0; i < N\_do; ++i)  
 calc\_pi(N);  
  
 end\_cpu = clock();  
 gettimeofday(&end\_time, 0);  
  
 long time\_seconds = end\_time.tv\_sec - begin\_time.tv\_sec;  
 long time\_microseconds = end\_time.tv\_usec - begin\_time.tv\_usec;  
 double time\_elapsed = time\_seconds + time\_microseconds\*1e-6;  
 long time\_elapsed\_mcs = (long)(time\_elapsed\*1000000 + 0.5);  
  
 double cputime\_spent = (double)(end\_cpu - begin\_cpu) / CLOCKS\_PER\_SEC;  
 long cputime\_spent\_mcs = (long)(cputime\_spent\*1000000 + 0.5);  
  
 printf("CPU time spent: %f sec (%ld us)\n", cputime\_spent, cputime\_spent\_mcs);  
 printf("Real time spent: %f sec (%ld us)\n", time\_elapsed, time\_elapsed\_mcs);  
}  
  
float calc\_pi(unsigned N\_iters)  
{  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
  
 // #pragma omp parallel num\_threads(5051)  
 #pragma omp parallel  
 {  
 unsigned th\_n = omp\_get\_num\_threads();  
 unsigned th\_i = omp\_get\_thread\_num();  
 unsigned iter\_per\_th = N\_iters / th\_n;  
 unsigned lb = iter\_per\_th\*th\_i;  
 unsigned ub = 5051;  
 if(th\_i == th\_n-1)  
 ub = N\_iters-1;  
 else  
 ub = lb + iter\_per\_th-1;  
  
 float pi\_local = 0.0;  
  
 ALIGNED\_(32) float vres[8];  
 \_\_m256 onem = \_mm256\_set1\_ps(1.0);  
 \_\_m256 Nm = \_mm256\_set1\_ps(N\_f);  
 \_\_m256 buffm;  
  
 for(unsigned i = lb; i <= ub; i+=8)  
 {  
 float j = (float)i + 0.5;  
 buffm = \_mm256\_set\_ps(j, j+1.0, j+2.0, j+3.0, j+4.0, j+5.0, j+6.0, j+7.0);  
 buffm = \_mm256\_div\_ps(buffm, Nm);  
 buffm = \_mm256\_mul\_ps(buffm, buffm);  
 buffm = \_mm256\_add\_ps(buffm, onem);  
 buffm = \_mm256\_div\_ps(onem, buffm);  
  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 buffm = \_mm256\_hadd\_ps(buffm, buffm);  
 \_mm256\_store\_ps(vres, buffm);  
 pi\_local += vres[0] + vres[7];  
 }  
  
 #pragma omp atomic  
 pi += pi\_local;  
 }  
  
 pi \*= 4.0;  
 pi /= N\_iters;  
 // printf("%.10lf\n", pi);  
  
 return pi;  
}

Листинг lab1\_4.c:

/\*No optimization\*/  
/\*Data parallelism\*/  
/\*Thread parallelism\*/  
/\*Thread parallelism\*/  
/\*inline assembly\*/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <time.h>  
#include <sys/time.h>  
#include <malloc.h>  
  
#include <omp.h>  
  
#if defined(\_MSC\_VER)  
#define ALIGNED\_(x) \_\_declspec(align(x))  
#else  
#if defined(\_\_GNUC\_\_)  
#define ALIGNED\_(x) \_\_attribute\_\_ ((aligned(x)))  
#endif  
#endif  
  
  
#define N\_do 1000  
#define N 10000000 // KPATHO 8  
#define N\_do\_hotter 100  
#define N\_hotter 10000 // KPATHO 8  
  
float calc\_pi(unsigned N\_iters);  
  
int main()  
{  
 clock\_t begin\_cpu, end\_cpu;  
 struct timeval begin\_time, end\_time;  
  
 /\*PA3OrPEB\*/  
 for(unsigned i = 0; i < N\_do\_hotter; ++i)  
 calc\_pi(N\_hotter);  
  
 gettimeofday(&begin\_time, 0);  
 begin\_cpu = clock();  
  
 for(unsigned i = 0; i < N\_do; ++i)  
 calc\_pi(N);  
  
 end\_cpu = clock();  
 gettimeofday(&end\_time, 0);  
  
 long time\_seconds = end\_time.tv\_sec - begin\_time.tv\_sec;  
 long time\_microseconds = end\_time.tv\_usec - begin\_time.tv\_usec;  
 double time\_elapsed = time\_seconds + time\_microseconds\*1e-6;  
 long time\_elapsed\_mcs = (long)(time\_elapsed\*1000000 + 0.5);  
  
 double cputime\_spent = (double)(end\_cpu - begin\_cpu) / CLOCKS\_PER\_SEC;  
 long cputime\_spent\_mcs = (long)(cputime\_spent\*1000000 + 0.5);  
  
 printf("CPU time spent: %f sec (%ld us)\n", cputime\_spent, cputime\_spent\_mcs);  
 printf("Real time spent: %f sec (%ld us)\n", time\_elapsed, time\_elapsed\_mcs);  
}  
  
float calc\_pi(unsigned N\_iters)  
{  
 const float N\_f = (float)N\_iters;  
 float pi = 0.0;  
  
 // #pragma omp parallel num\_threads(5051)  
 #pragma omp parallel  
 {  
 unsigned th\_n = omp\_get\_num\_threads();  
 unsigned th\_i = omp\_get\_thread\_num();  
 unsigned iter\_per\_th = N\_iters / th\_n;  
 unsigned lb = iter\_per\_th\*th\_i;  
 unsigned ub = 5051;  
 if(th\_i == th\_n-1)  
 ub = N\_iters-1;  
 else  
 ub = lb + iter\_per\_th-1;  
  
 float pi\_local = 0.0;  
  
 ALIGNED\_(32) float vres[8];  
  
 // float \*v8 = (float\*)aligned\_alloc(32, 8\*sizeof(float));  
 // float \*onev = (float\*)aligned\_alloc(32, 8\*sizeof(float));  
 // float \*Nm = (float\*)aligned\_alloc(32, 8\*sizeof(float));  
 ALIGNED\_(32) float v8[8]; ALIGNED\_(32) float onev[8]; ALIGNED\_(32) float Nm[8];  
 onev[0] = 1.0; onev[1] = 1.0; onev[2] = 1.0; onev[3] = 1.0;  
 onev[4] = 1.0; onev[5] = 1.0; onev[6] = 1.0; onev[7] = 1.0;  
 Nm[0] = N\_f; Nm[1] = N\_f; Nm[2] = N\_f; Nm[3] = N\_f;  
 Nm[4] = N\_f; Nm[5] = N\_f; Nm[6] = N\_f; Nm[7] = N\_f;  
  
 for(unsigned i = lb; i <= ub; i+=8)  
 {  
 float j = (float)i + 0.5;  
 v8[0] = j ; v8[1] = j+1.0; v8[2] = j+2.0; v8[3] = j+3.0;  
 v8[4] = j + 4.0; v8[5] = j+5.0; v8[6] = j+6.0; v8[7] = j+7.0;  
 \_\_asm\_\_ \_\_volatile\_\_(  
 "vmovaps (%[v8]), %%ymm0 \n\t"  
 "vmovaps (%[Nm]), %%ymm1 \n\t"  
 "vmovaps (%[onev]), %%ymm2 \n\t"  
 "vdivps %%ymm1, %%ymm0, %%ymm0 \n\t"  
 "vmulps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vaddps %%ymm0, %%ymm2, %%ymm0 \n\t"  
 "vdivps %%ymm0, %%ymm2, %%ymm0 \n\t"  
 "vhaddps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vhaddps %%ymm0, %%ymm0, %%ymm0 \n\t"  
 "vmovaps %%ymm0, %[res] \n\t"  
 : [res] "=m"(\*vres)  
 : [v8] "r"(v8), [Nm] "r"(Nm), [onev] "r"(onev)  
 : "%ymm0", "%ymm1", "%ymm2"  
 );  
 pi\_local += vres[0] + vres[7];  
 }  
  
 #pragma omp atomic  
 pi += pi\_local;  
   
 // free(v8); free(onev); free(Nm);  
 }  
  
 pi \*= 4.0;  
 pi /= N\_iters;  
 // printf("%.10lf\n", pi);  
  
 return pi;  
}