Национальный исследовательский ядерный университет «МИФИ»
(Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Отчёт
по результатам выполнения
Лабораторной работы №9
«Контракты с компилятором»

Дисциплина:	Практические Аспекты Разработки Высокопроизводительного Программного Обеспечения (ПАРВПО)
Студент:	Гареев Рустам Рашитович
Группа:	Б22-505
Преподаватель:	Куприяшин Михаил Андреевич
Дата:	4.06.2025

Москва - 2025

Оглавление

Технологический стек

memory	8GiB Системная память
processor	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz
siblings	8
cpu cores	4
bridge	11th Gen Core Processor Host Bridge/DRAM Registers
display	TigerLake-LP GT2 [Iris Xe Graphics]
gcc	version 13.3.0
openmp	version 201511(4.5)
OC	Ubuntu 24.04.2 LTS
IDE	Visual Studio Code 1.98.2

Тестовый алгоритм

Чтобы обеспечить достаточно высокую нагрузку на процессор, был выбран алгоритм решения 50 000 000 квадратных уравнений вида:

$$a*x^2+b*x+c = 0,$$

$$a*x^2+b*x+c=0.$$

Коэффициенты a, b и c генерируются равномерно в диапазоне [−1000;1000] при помощи генератора mt19937_64 (seed = 42). Такой диапазон выбран эмпирически: он адекватно покрывает типичные соотношения величин, когда дискриминант D=b^2−4*a*c порой принимает отрицательные и порой положительные значения, обеспечивая равномерное распределение случаев «нет вещественных корней» и «два вещественных корня». Поскольку последовательность коэффициентов фиксирована, результаты каждого запуска можно сравнивать без дополнительных статистических погрешностей.

Каждое итерационное решение уравнения включает:

- вычисление дискриминанта D;

- если D < 0, переход на следующую итерацию;

- если D ≥ 0, вычисление корней и подсчёт уравнений, у которых есть действительные корни. При этом сами корни сохраняются только временно, их значения не выводятся, однако вычисление всегда выполняется, чтобы нагрузка оставалась стабильной.

Ниже приведён исходный код алгоритма, решающего поставленную задачу.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <random>
#include <chrono>
#include <fstream>

#define NUM 50'000'000
#define SEED 42
```

```
using namespace std;
using namespace chrono;

struct EquationResult {
double a, b, c;
int num_roots;
double root1, root2;
};

int solve_quadratic(double a, double b, double c, double &x1, double &x2) {
double d = b * b - 4 * a * c;
if (d > 0) {
double sqrt_d = sqrt(d);
x1 = (-b + sqrt_d) / (2 * a);
x2 = (-b - sqrt_d) / (2 * a);
return 2;
} else if (d == 0) {
x1 = x2 = -b / (2 * a);
return 1;
} else {
x1 = x2 = 0;
return 0;
}
}

int main() {
mt19937 rng(SEED);
uniform_real_distribution<double> dist(-1000.0, 1000.0);
vector<EquationResult> results;
results.reserve(NUM);
auto start = high_resolution_clock::now();
for (size_t i = 0; i < NUM; ++i) {
double a = dist(rng);
while (fabs(a) < 1e-6) a = dist(rng);
double b = dist(rng);
double c = dist(rng);
double x1, x2;
int num_roots = solve_quadratic(a, b, c, x1, x2);
results.push_back({a, b, c, num_roots, x1, x2});
}
auto end = high_resolution_clock::now();
double elapsed = duration<double>(end - start).count();
cout << "Solved " << NUM << " equations in " << elapsed << " seconds." << endl;
return 0;
}
```

Листинг 1 — Исходный код разработанного алгоритма

Результаты вычислительного эксперимента

Ниже приведена сводная таблица полученных средних значений (трёх прогонов) по каждому уровню оптимизации.

Таблица 1 — Сводная сравнительная таблица результатов вычислительного эксперимента (решение квадратных уравнений)

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-O0	BASELINE	var	OFF	0.470883
-O0	BASELINE	var	ON	0.116705
-O0	BASELINE	const	OFF	0.475072
-O0	BASELINE	const	ON	0.119075
-O0	CONST_ALL	var	OFF	0.486534
-O0	CONST_ALL	var	ON	0.126722
-O0	CONST_ALL	const	OFF	0.483323
-O0	CONST_ALL	const	ON	0.118675
-O0	VOLATILE_ALL	var	OFF	0.485202
-O0	VOLATILE_ALL	var	ON	0.116368
-O0	VOLATILE_ALL	const	OFF	0.485364
-O0	VOLATILE_ALL	const	ON	0.124180
-O0	RESTRICT_ALL	var	OFF	0.482740
-O0	RESTRICT_ALL	var	ON	0.122718
-O0	RESTRICT_ALL	const	OFF	0.484462
-O0	RESTRICT_ALL	const	ON	0.130085

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-O1	BASELINE	var	OFF	0.0522144
-O1	BASELINE	var	ON	0.0251048
-O1	BASELINE	const	OFF	0.0717778
-O1	BASELINE	const	ON	0.0258026
-O1	CONST_ALL	var	OFF	0.0658764
-O1	CONST_ALL	var	ON	0.0244069
-O1	CONST_ALL	const	OFF	0.0714630
-O1	CONST_ALL	const	ON	0.0252078

-O1	VOLATILE_ALL	var	OFF	0.0868489
-O1	VOLATILE_ALL	var	ON	0.0374718
-O1	VOLATILE_ALL	const	OFF	0.0823535
-O1	VOLATILE_ALL	const	ON	0.0304658
-O1	RESTRICT_ALL	var	OFF	0.0735732
-O1	RESTRICT_ALL	var	ON	0.0243824
-O1	RESTRICT_ALL	const	OFF	0.0698553
-O1	RESTRICT_ALL	const	ON	0.0289920

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-O2	BASELINE	var	OFF	0.0484383
-O2	BASELINE	var	ON	0.0239759
-O2	BASELINE	const	OFF	0.0624644
-O2	BASELINE	const	ON	0.0269251
-O2	CONST_ALL	var	OFF	0.0573057
-O2	CONST_ALL	var	ON	0.0264781
-O2	CONST_ALL	const	OFF	0.0577907
-O2	CONST_ALL	const	ON	0.0379626
-O2	VOLATILE_ALL	var	OFF	0.0867538
-O2	VOLATILE_ALL	var	ON	0.0249516
-O2	VOLATILE_ALL	const	OFF	0.0829498
-O2	VOLATILE_ALL	const	ON	0.0263242
-O2	RESTRICT_ALL	var	OFF	0.0601380
-O2	RESTRICT_ALL	var	ON	0.0285015
-O2	RESTRICT_ALL	const	OFF	0.0671778
-O2	RESTRICT_ALL	const	ON	0.0325798

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-O3	BASELINE	var	OFF	0.0507948
-O3	BASELINE	var	ON	0.0232018
-O3	BASELINE	const	OFF	0.0551122
-O3	BASELINE	const	ON	0.0343237
-O3	CONST_ALL	var	OFF	0.0596695
-O3	CONST_ALL	var	ON	0.0299710
-O3	CONST_ALL	const	OFF	0.0647705

-O3	CONST_ALL	const	ON	0.0263798
-O3	VOLATILE_ALL	var	OFF	0.0840017
-O3	VOLATILE_ALL	var	ON	0.0240499
-O3	VOLATILE_ALL	const	OFF	0.0834127
-O3	VOLATILE_ALL	const	ON	0.0261219
-O3	RESTRICT_ALL	var	OFF	0.0579810
-O3	RESTRICT_ALL	var	ON	0.0342987
-O3	RESTRICT_ALL	const	OFF	0.0621913
-O3	RESTRICT_ALL	const	ON	0.0258989

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-Ofast	BASELINE	var	OFF	0.0486733
-Ofast	BASELINE	var	ON	0.0233148
-Ofast	BASELINE	const	OFF	0.0588285
-Ofast	BASELINE	const	ON	0.0272262
-Ofast	CONST_ALL	var	OFF	0.0633615
-Ofast	CONST_ALL	var	ON	0.0290869
-Ofast	CONST_ALL	const	OFF	0.0648039
-Ofast	CONST_ALL	const	ON	0.0283121
-Ofast	VOLATILE_ALL	var	OFF	0.0801266
-Ofast	VOLATILE_ALL	var	ON	0.0275918
-Ofast	VOLATILE_ALL	const	OFF	0.0823788
-Ofast	VOLATILE_ALL	const	ON	0.0258330
-Ofast	RESTRICT_ALL	var	OFF	0.0588697
-Ofast	RESTRICT_ALL	var	ON	0.0318459
-Ofast	RESTRICT_ALL	const	OFF	0.0621532
-Ofast	RESTRICT_ALL	const	ON	0.0252778

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-Og	BASELINE	var	OFF	0.313153
-Og	BASELINE	var	ON	0.066283
-Og	BASELINE	const	OFF	0.322102
-Og	BASELINE	const	ON	0.0763105
-Og	CONST_ALL	var	OFF	0.315117

-Og	CONST_ALL	var	ON	0.0733871
-Og	CONST_ALL	const	OFF	0.320342
-Og	CONST_ALL	const	ON	0.0885079
-Og	VOLATILE_ALL	var	OFF	0.309011
-Og	VOLATILE_ALL	var	ON	0.0737893
-Og	VOLATILE_ALL	const	OFF	0.308816
-Og	VOLATILE_ALL	const	ON	0.0841769
-Og	RESTRICT_ALL	var	OFF	0.318431
-Og	RESTRICT_ALL	var	ON	0.0756918
-Og	RESTRICT_ALL	const	OFF	0.319249
-Og	RESTRICT_ALL	const	ON	0.0801344

Компиляция	**Режим**	**Граница**	**OpenMP**	**Время (с)**
-Os	BASELINE	var	OFF	0.287595
-Os	BASELINE	var	ON	0.071174
-Os	BASELINE	const	OFF	0.297705
-Os	BASELINE	const	ON	0.0648385
-Os	CONST_ALL	var	OFF	0.318612
-Os	CONST_ALL	var	ON	0.0772107
-Os	CONST_ALL	const	OFF	0.317563
-Os	CONST_ALL	const	ON	0.0768522
-Os	VOLATILE_ALL	var	OFF	0.315659
-Os	VOLATILE_ALL	var	ON	0.0865852
-Os	VOLATILE_ALL	const	OFF	0.314155
-Os	VOLATILE_ALL	const	ON	0.0782784
-Os	RESTRICT_ALL	var	OFF	0.304936
-Os	RESTRICT_ALL	var	ON	0.0745648
-Os	RESTRICT_ALL	const	OFF	0.305094
-Os	RESTRICT_ALL	const	ON	0.0811949

Компиляция	Режим	Граница	OpenMP	Время (с)
-Oz	BASELINE	var	OFF	0.301222
-Oz	BASELINE	var	ON	0.0776135
-Oz	BASELINE	const	OFF	0.311627
-Oz	BASELINE	const	ON	0.0796629
-Oz	CONST_ALL	var	OFF	0.311877
-Oz	CONST_ALL	var	ON	0.0849346
-Oz	CONST_ALL	const	OFF	0.311870
-Oz	CONST_ALL	const	ON	0.0913983
-Oz	VOLATILE_ALL	var	OFF	0.318017
-Oz	VOLATILE_ALL	var	ON	0.0787287
-Oz	VOLATILE_ALL	const	OFF	0.316517
-Oz	VOLATILE_ALL	const	ON	0.0772910
-Oz	RESTRICT_ALL	var	OFF	0.296170
-Oz	RESTRICT_ALL	var	ON	0.0789196
-Oz	RESTRICT_ALL	const	OFF	0.294605
-Oz	RESTRICT_ALL	const	ON	0.0903484

Таблица 2 — Сводная сравнительная таблица результатов вычислительного эксперимента (вычисление «бесполезной» суммы)

Компиляция	Время (sum non-volatile), с	Время (sum volatile), с
-O0	0.431334	0.436279
-O1	0.392448	0.446375
-O2	0.197630	0.446905
-O3	0.197815	0.457334
-Ofast	0.109547	0.445445
-Og	0.391175	0.447779
-Os	0.193773	0.440291
-Oz	0.194406	0.442284

Обсуждение результатов

В ходе эксперимента была исследована производительность однородного вычислительного цикла (решение 50 000 000 квадратных уравнений) и вспомогательной задачи «бесполезной суммы» в зависимости от трёх факторов: уровня оптимизации компилятора (диапазон флагов от –O0 до –Oz), применения контрактов (const, volatile, restrict) и использования параллельного выполнения через OpenMP. Основной алгоритм решения квадратных уравнений оставался неизменным: генерация случайных коэффициентов, вычисление дискриминанта и, при необходимости, корней. Проверка «бесполезной суммы» была реализована в виде обычного прохода по большому массиву и суммирования элементов, при этом в одном случае результирующая переменная объявлялась volatile, а в другом — без этого модификатора.

Влияние уровня оптимизации

При сборке без оптимизаций (-O0) решение квадратных уравнений в однопоточном режиме (BASELINE_NOOMP, граница цикла «var») обрабатывается за ≈ 0.47 с, тогда как при включении минимальных («дебаг-ориентированных») оптимизаций -Og время примерно соответствует -O0 (≈ 0.31 с). Начиная с -O1, производительность резко возрастает: уже при -O1 однопоточный запуск занимает ≈ 0.052 с, а при -O2/-O3 — ≈ 0.049…0.051 с. Флаг -Ofast даёт незначительное дополнительное улучшение (~ 0.048 с). Оптимизации по размеру (-Os, -Oz) работают чуть медленнее, но всё равно значительно быстрее, чем -O0. Таким образом, глубина оптимизации компилятора оказывает критически важное влияние на скорость выполнения вычислительного цикла: разница между -O0 и -O2/-O3 достигает порядка 10x.

Влияние ключевых слов const, volatile, restrict на производительность

Когда мы объявляем лишь границу цикла как const (режим BASELINE с bound=const), однопоточная производительность при -O2 ухудшается чуть заметно: время растёт с ≈ 0.048 с до ≈ 0.062 с. Это говорит о том, что в данном конкретном коде компилятор, даже без наших подсказок, уже смог вывести «постоянство» предела (борьба с неопределённостью минимальна), а принудительная декларация const в некоторых случаях заставляет генерировать менее эффективный код ветвления.

В режиме CONST_ALL (все неизменяемые переменные объявлены как const) однопоточный запуск при -O2 показывает результат ≈ 0.057 с (смещение в сторону ухудшения по сравнению с BASELINE). Существенной выгоды за счёт повсеместного const не обнаружено: компилятор уже сам эффективно инлайнит и кэширует неизменяемые значения.

При объявлении всех переменных как volatile (режим VOLATILE_ALL) производительность на -O2 падает более заметно: ≈ 0.087 с в однопоточном режиме. Это естественно, поскольку ключевое слово volatile запрещает оптимизирующий компилятор хранить переменные в регистрах и кэшировать их между итерациями, вынуждая каждую операцию доступа обращаться в память. В многопоточном (OpenMP) режиме задержка «volatile» не столь велика, но всё равно заметна: VOLATILE_ALL_OMP ≈ 0.025 с (по сравнению с ≈ 0.024 с у BASELINE_OMP).

Режим RESTRICT_ALL (всех указателей объявляем restrict) в наших экспериментах дал лишь незначительные выигрыши или немного ухудшил производительность по сравнению с BASELINE. При -O2 однопоточный RESTRICT_ALL_NOOMP — ≈ 0.060 с (против 0.048 с у BASELINE_NOOMP); а многопоточный (RESTRICT_ALL_OMP) — ≈ 0.0285 с (против 0.02398 с у BASELINE_OMP). Полученный эффект минимален, поскольку стандартная реализация цикла не создавала конфликтов по указателям: компилятор и так уже умел выводить отсутствие перекрытия, а restrict тут лишь формальная подсказка, которая в данном коде не позволила убрать каких-либо значимых проверок.

Влияние OpenMP

Во всех режимах (BASELINE, CONST_ALL, VOLATILE_ALL, RESTRICT_ALL) многопоточный запуск (OpenMP = ON) примерно в 2…4 раза быстрее однопоточного при прочих равных условиях. Например, при -O2 BASELINE_NOOMP занимает ≈ 0.048 с, а BASELINE_OMP — ≈ 0.024 с, то есть ускорение порядка 2x. С ростом уровня оптимизации разница между ON/OFF стабильно сохраняется на уровне 2x. Это говорит о том, что операция решения уравнений хорошо распараллеливается, а накладные расходы OpenMP (инициализация поточного пула, синхронизация) при таких объёмах задач становятся невелики.

При -O0 мы наблюдаем искажение: однопоточный код выполняется очень медленно (≈ 0.47 с), а OpenMP уже даёт ускорение до ≈ 0.12 с — то есть более

чем 4 раза. На уровне без оптимизаций каждый прямой доступ к глобальным данным и к границе цикла выполняется медленно, а многопоточное ветвление позволяет частично скрыть эти накладные расходы благодаря распределению нагрузки.

«Бесполезная» сумма

Фиксация суммы как обычной (non-volatile) при -O2 показала время ≈ 0.197 с, а то же вычисление с переменной volatile — 0.447 с, то есть падение производительности > 2x. При -O3/-Os/-Oz независимость разницы сохраняется: суммы non-volatile выполняются примерно в 0.19 с, а volatile — ≈ 0.44 с.

Это наглядно подтверждает, что компилятор при наличии volatile не может оптимизировать ни один шаг суммирования: каждая запись и чтение вынуждены происходить через память и не выносятся в регистры. При оптимизируемом (non-volatile) коде компилятор часто разворачивает цикл (loop unrolling) или сохраняет частичный аккумулятор в регистре, сводя к минимуму обращения к памяти.

Выводы

1. Ключевые слова const и restrict в данном сценарии не дали заметного выигрыша, а в ряде случаев даже слегка ухудшили производительность, поскольку компилятору и без нашей подсказки уже было достаточно информации для эффективной оптимизации. В общем случае передача «обещаний» через const или restrict становится критичной лишь тогда, когда без них компилятор не может справиться — например, в более сложных сценариях с динамически выделенными массивами или многомерными указателями. Здесь же проверка кода оказалась тривиальной, и компилятор справляется сам.

2. Ключевое слово volatile при любых флагах значительно снижает производительность: в цикле решения квадратных уравнений падение времени в однопоточном режиме достигает почти 2x (при -O2) или даже более (при -O0), а вычисление «бесполезной суммы» с volatile против non-volatile замедлялось в 2-2.5 раза. Это демонстрирует, что volatile действительно отключает большинство оптимизаций «по хранению в регистрах» и вынуждает каждое обращение обращаться в память, что сильно удорожает даже простейшие арифметические циклы.

3. Параллельное выполнение OpenMP даёт устойчивый выигрыш ~ 2–4 раза в зависимости от уровня оптимизации. При высокой оптимизации компиляции (-O2, -O3, -Ofast) накладные расходы OpenMP (создание и синхронизация потоков) уже минимальны, поэтому распараллеливание цикла даёт почти идеальное деление работы. Это подтверждает, что для тяжёлых однородных вычислительных циклов применение OpenMP является наиболее эффективным способом ускорения, если аппаратные ресурсы это позволяют.

4. Уровень оптимизации компилятора оказался решающим фактором для высокой производительности. Без оптимизаций (-O0, -Og) вычислительный цикл выполнялся на порядки медленнее: более 0.4 с против 0.05 с при -O2–-O3. Расширенные оптимизации (-Ofast) давали лишь небольшое улучшение по сравнению с -O3 (~ 0.048 с против 0.050 с), что свидетельствует о том, что к настоящему моменту большинство «тривиальных» оптимизаций уже включены в уровень -O3. Оптимизации по размеру (-Os, -Oz) работали медленнее, чем -O2/-O3, разница была существенна (0.29-0.32 с против 0.048 с для обычного кода без OpenMP), что указывает: уменьшенный размер бинарника жертвует скоростью.

5. Лучшая комбинация в тестах — это -O2 (или -O3) + OpenMP + без ключевого volatile. Именно она дала минимальное время решения (~ 0.023 с) и не вносила дополнительных накладных расходов, связанных с «обещаниями компилятору». В то же время, применение volatile либо снижало пользу от оптимизаций, либо полностью нивелировало их, что подчёркивает: ключевое слово volatile следует использовать исключительно там, где это действительно необходимо (доступ к аппаратным регистраторам, межпроцессное взаимодействие и т. п.).

Таким образом, эксперимент подтвердил, что самая важная оптимизация в C/C++ — это правильный выбор уровня компиляции; ключевые контракты const/restrict стоит применять тогда, когда без них компилятору действительно не хватает информации о безопасности оптимизаций; а volatile «выключает» большинство оптимизаций и резко замедляет вычисления. Наконец, OpenMP остаётся простой и эффективной технологией для распараллеливания вычислительных циклов, особенно на многопроцессорных системах.

Ссылка на гит-репоизторий : **https://github.com/sagilyp/PAPHSD-2/tree/main/lab9**