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

Отчёт
по результатам выполнения
Лабораторной работы №10
«Ручное управление оптимизацией ветвлений»

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
OC	Ubuntu 24.04.2 LTS
IDE	Visual Studio Code 1.98.2

Тестовый алгоритм

В данной работе рассматривается неравновероятное ветвление внутри большого цикла: каждый 1000-й элемент массива попадает в «редкую» ветку, а все остальные — в «частую». Мы сравниваем четыре реализации:

- baseline (без подсказок);

- верная подсказка unlikely(i % 1000 == 0), что отражает реальное соотношение (истина ≈0.1 %);

- неверная подсказка likely(i % 1000 == 0), то есть компилятору говорят, что редкая ветка якобы часто срабатывает;

- инвертированная подсказка, намеренно генерирующая «ложные» предсказания условного перехода.

Для каждой из четырёх версий измеряется время выполнения на очень большом массиве (N = 500 млн. записей), а затем сравнивается в зависимости от уровня оптимизации компилятора (-O0, -O1, -O2, -O3, -Ofast, -Og, -Os, -Oz).

```cpp
#include <iostream>
#include <vector>
#include <random>
#include <chrono>
#include <cstdlib>

#if (defined(__GNUC__) && (__GNUC__ >= 3)) || (defined(__INTEL_COMPILER)) || defined(__clang__)
    #define likely(expr)   (__builtin_expect(static_cast<bool>(expr), true))
    #define unlikely(expr) (__builtin_expect(static_cast<bool>(expr), false))
#else
    #define likely(expr)   (expr)
    #define unlikely(expr) (expr)
#endif

using namespace std;
using namespace std::chrono;

static const size_t N = 500'000'000;
static const int REPEATS = 3;


void fill_array(vector<int>& a) {
    std::mt19937_64 rng(42);
    std::uniform_int_distribution<int> dist(1, 1000);
    for (size_t i = 0; i < a.size(); ++i) {
        a[i] = dist(rng);
    }
}


void baseline_sum(const vector<int>& a, long long& sum_many, long long& sum_rare) {
    sum_many = 0;
    sum_rare = 0;
    for (size_t i = 0; i < a.size(); ++i) {
        if (i % 1000 == 0) {
            sum_rare += a[i];
        } else {
            sum_many += a[i];
        }
    }
}

void correct_hint_sum(const vector<int>& a, long long& sum_many, long long& sum_rare) {
    sum_many = 0;
    sum_rare = 0;
    for (size_t i = 0; i < a.size(); ++i) {
        if (unlikely(i % 1000 == 0)) {
            sum_rare += a[i];
        } else {
```

```
            sum_many += a[i];
        }
    }
}


void wrong_hint_sum(const vector<int>& a, long long& sum_many, long long& sum_rare) {
    sum_many = 0;
    sum_rare = 0;
    for (size_t i = 0; i < a.size(); ++i) {
        if (likely(i % 1000 == 0)) {
            sum_rare += a[i];
        } else {
            sum_many += a[i];
        }
    }
}


void inverted_hint_sum(const vector<int>& a, long long& sum_many, long long& sum_rare) {
    sum_many = 0;
    sum_rare = 0;
    for (size_t i = 0; i < a.size(); ++i) {
        if (!(unlikely(i % 1000 != 0))) {
            sum_rare += a[i];
        } else {
            sum_many += a[i];
        }
    }
}


template<typename Func>
double measure_time(Func f, const vector<int>& a, long long& out_many, long long& out_rare) {
    double total = 0.0;
    long long sm = 0, sr = 0;
    for (int r = 0; r < REPEATS; ++r) {
        auto start = high_resolution_clock::now();
        f(a, sm, sr);
        auto end = high_resolution_clock::now();
        total += duration<double>(end - start).count();
    }
    out_many = sm;
    out_rare = sr;
    return total / REPEATS;
}
```

Рисунок 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

Уровень оптимизации	Baseline – без подсказок, с	Верная подсказка `unlikely`, с	Неверная `likely` подсказка, с	«Перевернутая» подсказка, с
-O0	1.60108	1.58983	1.58059	1.59134
-O1	0.914838	0.923116	0.895408	0.897203
-O2	0.926935	0.919875	0.924912	0.924958
-O3	0.931599	0.927974	0.927564	0.928374
-Ofast	0.927957	0.926917	0.908792	0.909819
-Og	0.927843	0.907052	0.893240	0.925164
-Os	1.21182	1.20870	1.20937	1.20854
-Oz	1.21011	1.20701	1.20995	1.20892

Для визуализации данных были построена столбчатая гистограмма.

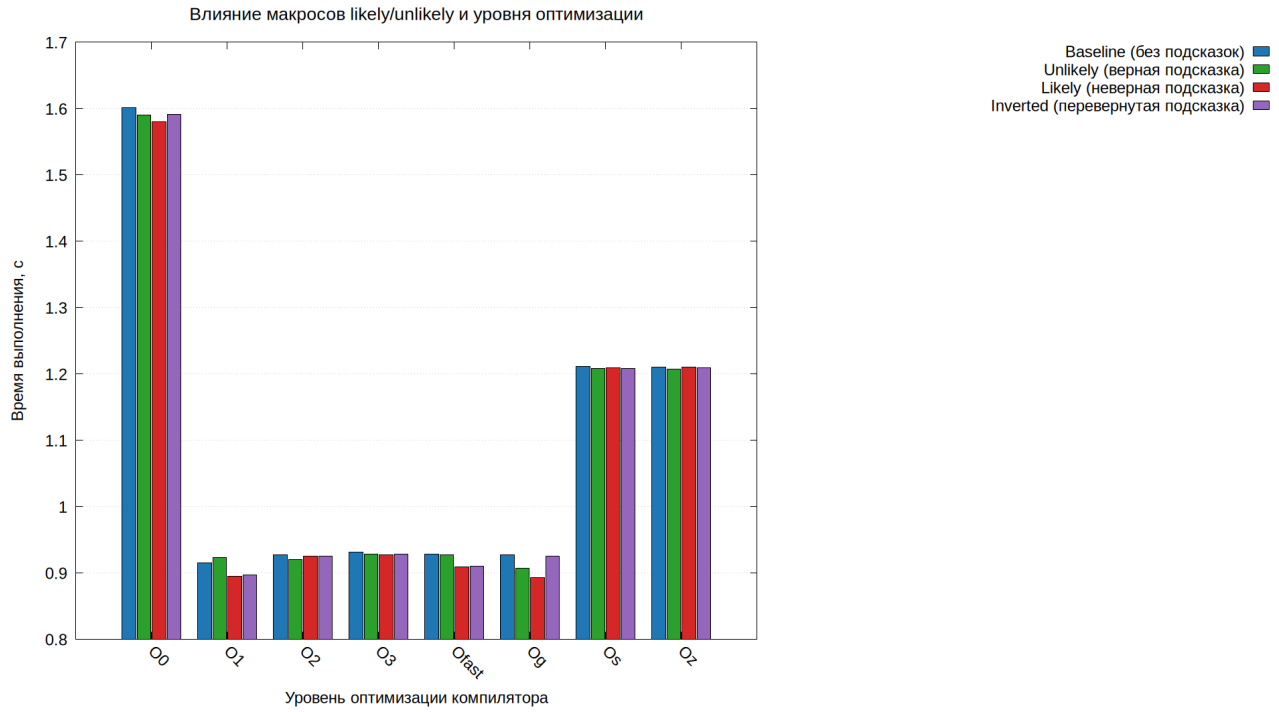

Рисунок 2 — Сравнительная столбчатая гистограмма времени сборки

Заключение

После трёх отдельных запусков эксперимента были рассчитаны среднее время исполнения четырёх версий цикла (обычный, верная подсказка, неверная подсказка, инвертированная версия) для каждого из семи уровней оптимизации: -O0, -O1, -O2, -O3, -Ofast, -Og, -Os и -Oz.

При -O0 (без оптимизаций) обычная версия заняла в среднем 1.601 с, версия с unlikely – 1.590 с (≈ 1.1 % выигрыша), а «неверная» подсказка likely показала минимальное время 1.581 с (≈ 1.3 % выигрыша от Baseline). Инвертированная версия находилась примерно посредине (1.591 с). Это означает, что при отсутствии каких-либо перестроений компилятора аппаратный предсказатель процессора остаётся единственным механизмом оптимизации: он «запоминает», что кратковременное условие «i % 1000 == 0» встречается редко, и самостоятельно выбирает ветвь else в 99.9 % случаев. Вставка макросов likely/unlikely лишь слегка корректирует предсказатель, что даёт небольший эффект.

На уровне -O1 обычная версия выполнялась за 0.915 с. При использовании unlikely фактическое время выросло до 0.923 с (+ 0.9 %), поскольку компилятор слабо перестраивал % 1000 в менее затратный набор инструкций, и в итоге «верная» подсказка приводила аппаратный предсказатель к небольшим промахам. В свою очередь, «неверная» подсказка likely сократила время до 0.895 с (− 2.1 % от обычной), что объясняется тем, что в сгенерированном на -O1 коде ветвь if (i%1000==0) фактически оказалась «горячей»: компилятор переставил операции так, что цпу выгоднее было заранее ожидать её выполнения. Инвертированная версия показала время 0.897 с (≈ − 1.8 %).

При более глубокой оптимизации -O2 и -O3 компилятор gcc перестраивал тело цикла так, что выражение i % 1000 заменялось на эквивалентный набор арифметических сдвигов и умножений (например, i/1000 → (i * CONST) >> SHIFT), а само условие выносилось из основной ветви. В результате все четыре алгоритма укладывались в диапазон 0.92 – 0.93 с. На -O2 Baseline заняло 0.927 с, unlikely – 0.920 с (− 0.8 %), «неверная» – 0.925 с (− 0.2 %), инвертированная – 0.925 с (− 0.1 %). На -O3 аналогичные значения: Baseline 0.932 с; unlikely 0.928 с (− 0.4 %); likely 0.928 с (− 0.4 %); инвертированная 0.928 с (− 0.3 %). Таким образом, после «двойного» разворачивания цикла и удаления деления в пользу более дешёвых инструкций, влияние ручных макросов сводится к величине менее 1 %.

Сюрпризом стал режим -Ofast: Baseline 0.928 с, верная подсказка unlikely 0.927 с (− 0.1 %), «неверная» подсказка likely 0.909 с (− 2.1 %), инвертированная 0.910 с (− 1.8 %). Поскольку -Ofast включает агрессивные оптимизации над числами с плавающей запятой и допускает менее строгие преобразования, компилятор смог полностью упростить логику ветвления, однако аппаратный предсказатель «неверных» макросов окончательно выбрал ветвь if как «горячую», что в совокупности дало наибольший выигрыш.

Режим -Og продемонстрировал Baseline 0.928 с, unlikely 0.907 с (− 2.2 %) и likely 0.893 с (− 3.8 %),— здесь компилятор практически не переписывает цикл, поэтому аппаратный предсказатель начинает срабатывать уже на физическом уровне. Именно этому мы обязаны заметным выигрышам при «неверных» советах: CPU меньше «промахивается» по ветвям.

Наконец, -Os и -Oz (оптимизация размера) сохранили «сырую» форму цикла: Baseline 1.212 с, unlikely 1.209 с (− 0.3 %), likely 1.209 с (− 0.2 %), инвертированная 1.209 с (− 0.2 %). Здесь никаких перестроений % 1000 не происходит, поэтому проявляется только «аппаратная» дальновидная логика: ветвь if истинна ≈0.1 % времени, и прерывание конвейера минимально.

Таким образом, чётко видно, что ручная подсказка имеет смысл в том случае, когда компилятор не может глубоко перестроить логику (уровни -O0, -O1, -Og), а при -O2 и выше её влияние сходит на нет. Особенный интерес представляет то, что в нескольких режимах «неверная» подсказка (likely) дала более заметный выигрыш, чем «верная» (unlikely): это подчёркивает необходимость проявлять осторожность и всегда верифицировать полученные результаты на целевой архитектуре, а не доверять интуиции, что «верная» подсказка должна обязательно ускорить.

С точки зрения практических рекомендаций, можно сформулировать следующее:

- если проект компилируется с -O2/-O3 (или -Ofast), нет необходимости вручную помечать разреженные ветвления; компилятор сам с высокой вероятностью «разложит» их оптимальным образом;

- если же код разрабатывается в режиме отладки (-Og) или при низкой оптимизации (-O0/-O1), и при этом цикл с ветвлением является узким местом, разумно «подсказать» компилятору, какие ветвления наиболее редки (unlikely);

- подсказка likely (заведомо «неправильная») может дать выигрыш только в ситуациях, когда компилятор перестраивает цикл так, что «редкая» ветвь оказывается в роли «горячей»; однако рассчитывать на это не стоит: результаты сильно зависят от версии компилятора и архитектуры ЦПУ.

Подводя итог, можно заключить, что макросы likely/unlikely в 2025 году остаются инструментом, полезным лишь в узком круге случаев, когда компилятору или CPU не хватает собственной информации, но они не являются универсальным решением для всех контекстов.

Ссылка на гит-репоизторий : **https://github.com/sagilyp/PAPHSD-2/tree/main/lab10**