Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий
Высшая школа программная инженерия

Лабораторная работа №4

Тема: Определение опций оптимизации, для приложения

Работу выполнил студент
_____ А. С. Авферонок
1 курс группа № в13534/22
«___» _____ 2019 г.

Преподователь
_____ А. В. Петров
«___» _____ 2019 г.

Санкт-Петербург, 2019 г.

Постановка задачи

Цель работы - написать сценарий, который будет компелировать программу с разными уровнями оптимизации, вычисление времени работы программы и вычисление занимаемого исполняемым файлом дискового пространства. Сценарий должен принемать имя исходного файла, а время работы исполняемого файла должно быть больше 20 сейнд.

Ход выполнения работы

Для выявления лучшей оптимизации необходимо было написать программу на языке C. С условием, что время работы исполняемого файла будет больше 20 секунд. Было выбранно приложение со следующим исходным кодом:

```
    #include <stdio.h>
    double powern (double d, float n) {
        double x = 1.0;
unsigned j;
for (j = 1; j <= n; j++)
    x *= d;
return x;
    }

    int main (void) {
double sum = 0.0;
float i;
for (i = 1; i <= 1000000; i=i+10)
{
    sum += powern (i, i / 5);
}
printf ("sume = %g\n", sum);
return 0;
    }
```

Время работы его исполняемого файла составли 24,357 секнды, что удовлетворяет условию из постановки задачи.

Далее необходимо было составить сценарий для компеляции выбранного исходного кода и вывода результатов времени исполнения и размера исполняемого файла. Текст сценария предоставлен в листнинге.

Входе компеляций исходного файла были перебранны основные ключи компиляции O0, Os, O1, O2, O3, O2 -march=native, O3 -march=native, O2 -march=native -funroll-loops,O3 -march=native -funroll-loops. Результаты времени исполнения и размера исполняемого файла с использованием разных ключей представленны в таблице 1.

Таблица 1: Результат компиляций с основными ключами

Ключ оптимизации	Время исполнения, сек	Размер, байт
O0	24,357	8672
Os	10,940	8672
O1	10,711	8672
O2	10,693	8672
O3	10,615	8672
O2 -march=native	10,588	8672
O3 -march=native	10,697	8672
O2 -march=native -funroll-loops	11,026	8672
O3 -march=native -funroll-loops	10,662	8672

Было выявленно, что исполняемы фаил полученный в ходе компиляции с использованием ключа оптимизации O2 -march=native, исполняется быстрее всех, по этому он был выбран в качестве оптимальной опции.

Далее необходимо было выбрать провести оптимизацию с оптимальной опцией и межпроцедурной оптимезацией и оптимеацией времени компоновки. Результаты времени исполнения и размера исполняемого файла представленны в таблице 2.

Таблица 2: Результаты компиляций

Ключ оптимизации	Время исполнения, сек	Размер, байт
O2 -march=native -flto	10,749	8608
O2 -march=native -fipa-reference	10,595	8672

Было выявленно, что исполняемы фаил полученный в ходе компиляции с использованием ключа оптимизации O2 -march=native -flto, занимет меньше всех места на дисковом пространсве.

Далее необходимо было выбрать провести оптимизацию с оптимальной опцией и оптимизацией с обратной связью. Результаты времени исполнения и размера исполняемого файла представленны в таблице 3.

Таблица 3: Результаты компиляций

Ключ оптимизации	Время исполнения, сек	Размер, байт
O2 -march=native -fprofile-generate	10,601	28688
O2 -march=native -fprofile-use	10,558	8608

Было выявленно, что исполняемы фаил полученный в ходе компиляции с использованием ключа оптимизации O2 -march=native -fprofile-generate, занимет больше всех места на дисковом пространсве и он был исключен из испытаний, а компиляция с ключем O2 -march=native -fprofile-use дала прирост во времени исполнения.

Получив оптимальные опции оптимизации и обратной связью и межпроцедурной оптимизацией времени, было решено объеденить в едино для выяления наилучшего резултата. Результаты времени исполнения и размера исполняемого файла представленны в таблице 4.

Таблица 4: Результаты компиляций

Ключ оптимизации	Время исполнения, сек	Размер, байт
O2 -march=native -flto -fprofile-use	10,558	8608

Были выявлены наилучшие ключи оптимизации для компиляции выбранного исходного кода.

На рисунке 1 можно наглядно увидить соотношение времени работы исполняемого файла в зависимости от ключа оптимизации.

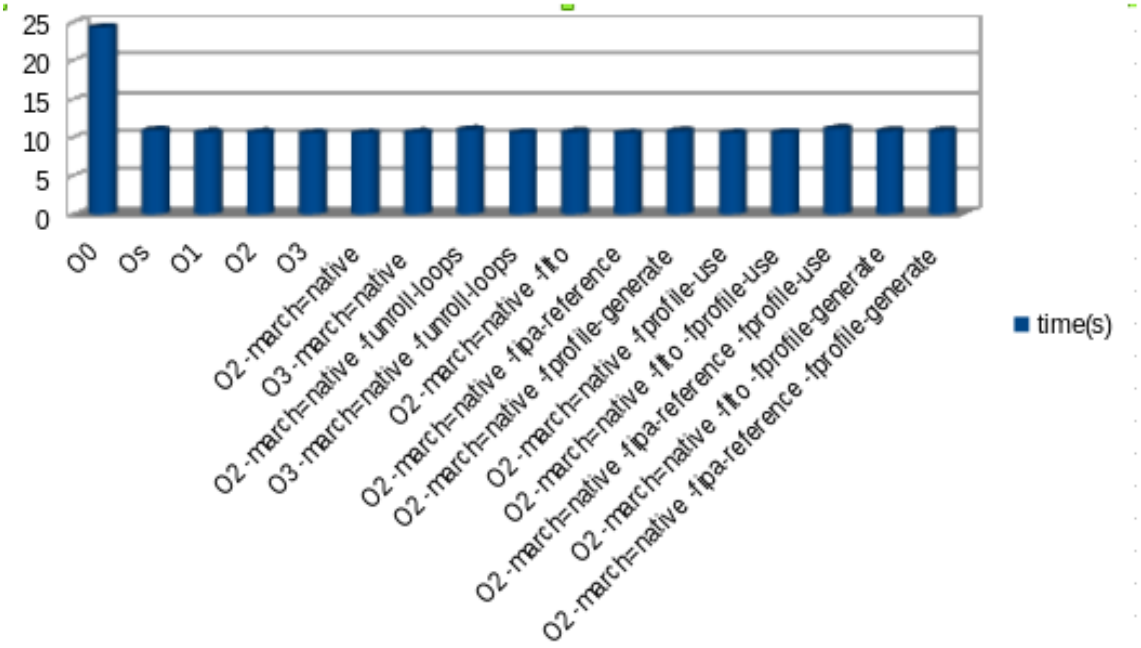

Рисунок 1. Отношение времени к ключу

На рисунке 2 можно наглядно увидить соотношение занимемого обмема на диске исполняемого файла в зависимости от ключа оптимизации.

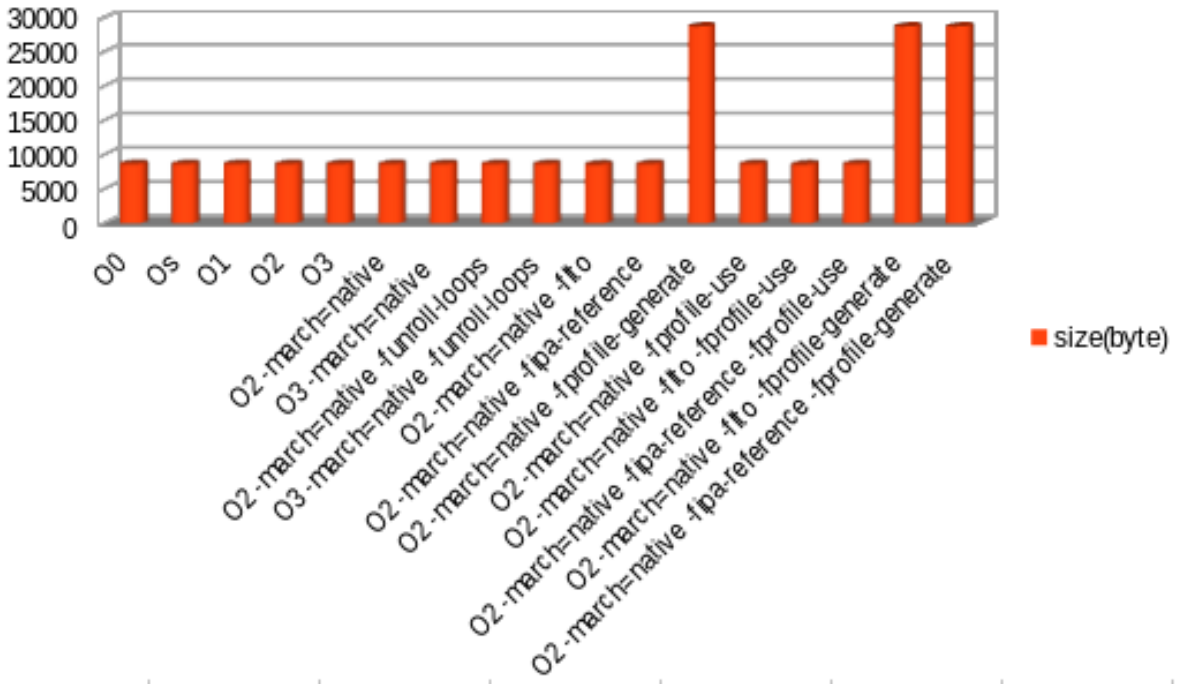

Рисунок 2. Отношение размера к ключу

Заключение

Вывод, в результате выполнения лабораторной работы было выявленно, что размер файла не изменялся до использования межпроцедурной оптимизацией, оптимизацией времени компоновки и оптимизацией с обратной связью. По этому ключ "O2 -march=native"исполнился за самый короткий промежуток, 10,588 секунд и он был выбран для дальнейшего спользования.

Исполняемый фаил занимает меньший объем дискового пространства используя ключ межпроцедурной оптимизацией "O2 -march=native -flto 8608 байт, а сильное увеличение размера файла получилось используя ключь оптимизации с обратной связью "O2 -march=native -fprofile-generate 28688 байт.

Но используя тоже ключ оптимизации с обратной связью "O2 -march=native -fprofile-use"программа исполнилась начительно быстрее, за 10,558 секунд при этом размер занимаемого дискового пространства файлом остался стандартным, 8672 байт.
Исходя из этого для достижения максимально оптимизированного решения было выбранно, сгруппировать ключи оптемизации в один.

В итоге, самым быстрм и малозатратным объема диска стал ключ "O2 -march=native -flto -fprofile-use время исполнения файла 10,558 секунды, размер занимаемого дискового пространства исполняемым файлом 8608 байт.

Листинг

Сценарий:

```
#!/bin/bash

name=$1

gcc -Wall -O0 $1
echo "-O0"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -Os $1
echo "-Os"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O1 $1
echo "-O1"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O2 $1
echo "-O2"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O3 $1
echo "-O3"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O2 -march=native $1
echo "-O2 -march=native"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"
```

```
gcc -Wall -O3 -march=native $1
echo "-O3 -march=native"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O2 -march=native -funroll-loops $1
echo "-O2 -march=native -funroll-loops"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O3 -march=native -funroll-loops $1
echo "-O3 -march=native -funroll-loops"
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "byte :$by"

gcc -Wall -O2 -march=native -flto $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -flto"
echo "byte :$by"

gcc -Wall -O2 -march=native -fipa-reference $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -fipa-reference"
echo "byte :$by"

gcc -Wall -O2 -march=native -fprofile-generate $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -fprofile-generate"
echo "byte :$by"

gcc -Wall -O2 -march=native -fprofile-use $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -fprofile-use"
echo "byte :$by"

gcc -Wall -O2 -march=native -flto -fprofile-use $1
out="$(time ./a.out)"
```

```
by="$(du -b a.out)"
echo "-O2 -march=native -flto -fprofile-use"
echo "byte :$by"

gcc -Wall -O2 -march=native -fipa-reference -fprofile-use $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -fipa-reference -fprofile-use"
echo "byte :$by"

gcc -Wall -O2 -march=native -flto -fprofile-generate $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -flto -fprofile-generate"
echo "byte :$by"

gcc -Wall -O2 -march=native -fipa-reference -fprofile-generate $1
out="$(time ./a.out)"
by="$(du -b a.out)"
echo "-O2 -march=native -fipa-reference -fprofile-generate"
echo "byte :$by"
```