Мерочкин Илья БПИ218

Индивидуальное домашнее задание по архитектуре вычислительных систем №3
Вариант 14

14. Разработать программу, вычисляющую с помощью степенного ряда с точностью не хуже 0.1% значение функции гиперболического котангенса cth(x) = $(e^{x} + e^{-x}) / (e^{x} - e^{-x})$ для заданного параметра x.

При x -> ∞, cth(x) -> 1, благодаря данному свойству, при x > 10 программа выводит 1. Аналогично при x < -10 программа выводит -1. Поэтому все основные вычисления происходят на отрезке [-10; 10].

4 - 6 баллов

Исходный код на языке C лежит в файле mark_6/asm.c, ассемблерная программа в файле mark_6/asm.s, ассемблерная с оптимизацией, за счет использование регистров процессора в файле mark_6/asm_registers.s.

Ассемблерная программа была откомпилирована без оптимизирующих и отладочных опций, добавлены комментарии, поясняющие эквивалентное представление на языке C. Это было сделано командой gcc -O0 -Wall -mask=intel -S asm.c -o asm.s.

Далее из ассемблерной программы были убраны лишние макросы, за счет использование аргументов командной строки и ручного редактирования исходного кода. Это было сделано командой gcc -masm=intel -fno-asynchronous-unwind-tables -fno-jump-tables -fno-stack-protector -fno-exceptions asm.c -S -o asm.s.

В реализованной программе используется функция с передачей данных через параметры, использованием локальных переменных и возвращаемым значением.

В ассемблерных программах соответственно добавлены комментарии, поясняющие эквивалентный код на C и эквивалентное использование регистров вместо переменных исходной программы на C.

Тестовые прогоны для трех программ:

Тест 1
Входные данные:
0.576
Выходные данные программы на C:
1.923994
Выходные данные программы на ассемблере:
1.923994
Выходные данные программы на ассемблере, с использованием регистров:
1.923994

Тест 2
Входные данные:
-1.234
Выходные данные программы на C:
-1.185205
Выходные данные программы на ассемблере:
-1.185205
Выходные данные программы на ассемблере, с использованием регистров:
-1.185205

Тест 3
Входные данные:
51
Выходные данные программы на C:
1.0000
Выходные данные программы на ассемблере:
1.0000
Выходные данные программы на ассемблере, с использованием регистров:
1.0000

Тест 4
Входные данные:
-15
Выходные данные программы на C:
-1.0000
Выходные данные программы на ассемблере:

-1.0000
Выходные данные программы на ассемблере, с использованием регистров:
-1.0000

На основе данных тестовых прогонов, все три программы корректно работают.

После рефакторинга программы с помощью регистров код стал более читаемые и интуетивно понятный, однако это не ускорило программу (объектные файлы, как в первом, так и во втором случае весят 16 kB).

7 баллов

Программа реализована в виде двух единиц компиляции.

asm.c, expon.c - файлы с кодом на C, asm.s, expon.s - файлы с ассемблерным кодом.

Изначально, файлы с кодом на C были отдельно откомпилированы в ассемблерные файлы, а затем, с помощью команды gcc asm.s expon.s -o ./asm скомпилированы в один исполняемый файл.

В программе присутствует файловый ввод/вывод. Имена файлов задаются с использованием аргументов командной строки. Всего вводятся два файлы: входной файл (в котором лежит число) и выходной файл. Программа проверяет на корректность число аргументов командной строки и корректное открытие файлов.

Подготовлены несколько файлов, обеспечивающих тестовое покрытие программы. В файлах input_testN лежат входные числа, а в файлах output_testN результаты выполнения программы на данных числах. Запускается программа следующим образом ./asm input_testN.txt output_testN.txt (./asm - исполняемый файл).

8 баллов

Формат ввода:

1) Чтобы сгенерировать данные рандомно, нужно просто запустить исполняемый файл, например, ./asm
2) Чтобы использовать консольный ввод/вывод нужно после исполняемого файла указать число, например, ./asm 1.5

3) Чтобы использовать файловый ввод/вывод после исполняемого файла нужно указать файл для ввода и файл для вывода, например, ./asm input.txt output.txt.

Время работы каждой программы замеряется и выводится в консоль (без учет ввода и вывода). Для большей точности при сравнении программа запускает поиск ответа 10000000 раз.

Таблица с примерами входных данных и времени работы:

Входное число	Время выполнения в миллисекундах
11	0
-15	0
1.5	1137
0.2	1113
-2	1164
-1.1	1132

Если исходное число меньше -10 или больше 10, то программа сразу выводит ответ (это было объяснено в самом начале), поэтому время выполнения для таких входных данных 0. Для чисел лежащих в диапазоне [-10; 10] программа делает 20 итераций, за которые гарантированно найдет точный ответ. Именно поэтому время работы программы для чисел в диапазоне [-10; 10] примерно одинаковое (т.к. программа делает константное число итераций).

9 баллов

Скомпилируем программу с использованием флага оптимизации -O3. Сделаем это командой gcc -S -O3 asm.c -o asm_optimizet.s и командой gcc -S -O3 expon.c -o expon_optimizet.s. Сразу заметно, что ассемблерные программа стали меньше, по сравнения с программами,

скомпилированными без оптимизирующего флага asm.s, expon.s (asm_optimizet.s меньше на 700 байт, а expon_optimizet.s на 400 байт).

Теперь получим исполняемый файл командой gcc asm_optimizet.s expon_optimizet.s -o ./asm_optimizet. Однако сам исполняемый файл asm_optimizet получился больше (правда всего на 100 байт).

Проведем замеры времени, и сравним его с работой программы без оптимизирующего ключа.

Входное число	Время работы программы без оптимизации	Время работы программы с оптимизацией
11	0	0
-15	0	0
1.5	1117	523
0.2	1116	510
-2	1144	518
-1.1	1141	519

Как видно, на всех тестах программа с оптимизацией работает быстрее в два раза, чем обычная.

Теперь сделаем все то же самое, но с флагом оптимизации -Os. Файл с флагом оптимизации asm_optimizem.s оказался на 1300 байт меньше файла без оптимизации asm.s, а expon_otimizem.s на 500 байт. Исполняемый же файл ./asm_optimizem оказался немного больше обычного.

Проведем замеры времени и сравним его для обеих программ.

Входное число	Время работы программы без оптимизации	Время работы программы с оптимизацией
11	0	0

-15	0	0
1.5	1117	2920
0.2	1116	2838
-2	1144	2874
-1.1	1141	2852

Как видно, во всех тестах, программа с флагом -Os работала сильно дольше изначальной.

Итог: флаг -Os сильно оптимизирует размер кода, однако также сильно повышает время выполнения программы. А флаг -O3 немного оптимизирует размер кода, но зато сильно уменьшает время выполнения программы.