Московский государственный университет им. М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра автоматизации систем вычислительных комплексов

Уменьшение времени выполнения программы

Фролов Александр
321 группа

Москва
2021

Содержание

1 Постановка задачи

- Измерить время выполнения при помощи утилиты time и ассемблерной инструкции rdtsc.
- Повторить замер 5 раз, взять среднее;
- Уменьшить время выполнения как можно больше, изменяя текст программы и настройки сборки;
- Собрать данные времени работы.

2 Конфигурация вычислительной системы

- Процессор Intel(R) Core(TM) i3-6006U CPU @ 2.00GHz с 4 ядрами;
- 12 ГБ оперативной памяти;
- Операционная система Ubuntu x64.

3 Результаты измерения времени

Время работы неоптимизированной программы, измеренное при помощи утилиты time и ассемблерной инструкции rdtsc:

	1	2	3	4	5	avg
real	0m20,796s	0m20,310s	0m20,649s	0m21,794s	0m20,663s	0m20,842s
user	0m20,757s	0m20,200s	0m20,493s	0m21,637s	0m20,642s	0m20,746s
sys	0m0,028s	0m0,093s	0m0,025s	0m0,020s	0m0,020s	0m0,037s
rdtsc	0m20,171s	0m20,148s	0m20,209s	0m20,099s	0m20,262s	0m20,178s

Замечание 1: время при помощи ассемблерной инструкции rdtsc считалось по формуле

$$time = \frac{E - S}{F_{max}}$$

где S и E - результат работы инструкции, вызванной в начале и в конце программы, F_{max} - максимальная частота процессора.

Замечание 2: замеры при помощи утилиты time и замеры при помощи инструкции rdtsc - разные пятерки замеров.

4 Способы ускорения выполнения программы и результаты их применения

Замечание: будем считать, что матрицы B и C не являются матрицами определенного вида, то есть их элементы - случайные действительные числа.

4.1 Оптимизация без нитей

1. Улучшение асимптотики алгоритма путем введения предподсчета сумм по строкам матрицы B и по столбцам матрицы C.

2. Во втором цикле for переменные i и j поменены местами для обеспечения последовательного доступа к памяти.

3. Использование более оптимального инкрементирования (++i).

4. Оптимизации компилятора (флаг -O3).

Время работы программы с данными оптимизациями:

	1	2	3	4	5	avg
real	0m0,064s	0m0,070s	0m0,041s	0m0,067s	0m0,068s	0m0,062s
user	0m0,025s	0m0,050s	0m0,037s	0m0,039s	0m0,044s	0m0,039s
sys	0m0,039s	0m0,020s	0m0,004s	0m0,027s	0m0,025s	0m0,023s
rdtsc	0m0,062s	0m0,061s	0m0,060s	0m0,062s	0m0,065s	0m0,062s

Таким образом, фактическое время вполнения программы было уменьшено примерно в 336 раз.

4.2 Оптимизация с нитями

Код, полученный при оптимизациях из пункта 4.1, был распараллелен при помощи библиотеки openMP. Оба внешних цикла параллелились отдельно. При компиляции использовалась та же оптимизация, что и в пункте 4.1 (-O3). Время работы программы с 4 нитями:

	1	2	3	4	5	avg
real	0m0,030s	0m0,031s	0m0,051s	0m0,029s	0m0,034s	0m0,035s
user	0m0,045s	0m0,061s	0m0,086s	0m0,047s	0m0,065s	0m0,061s
sys	0m0,053s	0m0,039s	0m0,083s	0m0,048s	0m0,046s	0m0,054s
rdtsc	0m0,045s	0m0,043s	0m0,043s	0m0,060s	0m0,052s	0m0,049s

Таким образом, фактическое время вполнения программы было уменьшено примерно в 600 раз в сравнении с первоначальным алгоритмом.

5 Методы оптимизации

5.1 Изменение асимптотики алгоритма

Было создано два массива: $b_{precalc}$ и $c_{precalc}$ - суммы по строкам матрицы B и суммы по столбцам матрицы C. Это позволило изменить асимптотику с $O(n^3)$ до $O(n^2)$.

5.2 Оптимизации компилятора

Оптимизации компилятора - методы, которые применяются компилятором для получения более оптимального программного кода при сохранении его функциональных возможностей.

Среди принципов оптимизации, применяемых в различных методах оптимизации в компиляторах (некоторые из них могут противоречить друг другу или быть неприменимыми при разных целях оптимизации):

- уменьшение избыточности: повторное использование результатов вычислений, сокращение числа перевычислений;
- компактификация кода: удаление ненужных вычислений и промежуточных значений;
- сокращение числа переходов в коде: например, использование встраивания функций или размотки цикла позволяет во многих случаях ускорить выполнение программы ценой увеличения размера кода;
- локальность: код и данные, доступ к которым необходим в ближайшее время, должны быть размещены рядом друг с другом в памяти, чтобы следовать принципу локальности ссылок;
- использование иерархии памяти: размещать наиболее часто используемые данные в регистрах общего назначения, менее используемые — в кэш, ещё менее используемые — в оперативную память, наименее используемые — размещать на диске.
- распараллеливание: изменение порядка операций может позволить выполнить несколько вычислений параллельно, что ускоряет исполнение программы.

Ключ $-O3$ применяет максимально возможное число оптимизаций, нацеленных на ускорение исполнения программы, в ущерб времени компиляции.

5.3 Последовательный доступ к памяти

Случайный доступ к памяти влечет за собой увеличение времени исполения программы, поэтому непоследовательный доступ к памяти сведен к минимуму.

5.4 Использование нитей

Нити позволяют параллельно выполнять некоторые участки кода, тем самым уменьшая время работы программы. Так как матрицы B и C не перезаписываются, пусть каждый из n потоков параллельно просчитает $\frac{SIZE}{n}$ различных строк матрицы A. Данный метод оптимизации наряду с оптимизациями компилятора дает наибольший выигрыш по скорости.

6 Вывод

Были использованы следующие методы оптимизации:

1. Снижение асимптотической сложности алгоритма.
2. Использование оптимизаций компилятора.
3. Использование многопоточная библиотека openMP.

Результаты представлены в таблице ниже:

	no opt	opt	goodness
real	0m20,842s	0m0,035s	595
user	0m20,746s	0m0,061s	340
rdtsc	0m20,178s	0m0,049s	412