BrozTeam

Параллельный алгоритм поиска общих подстрок, больших заданной длины

А. С. БОГАТЫЙ
Московский государственный университет имени М. В. Ломоносова
механико-математический факультет, 3 курс
e-mail: bogatyia@gmail.com
isn: pawnbot

И. С. БОГАТЫЙ
Московский государственный университет имени М. В. Ломоносова
механико-математический факультет, 5 курс
e-mail: bogatyi@gmail.com
isn: loken17

1 Введение

В конкурсе **Accelerate 2012** компании **Intel** была предложена задача поиска всех общих подстрок у двух строк, эти подстроки должны удовлетворять некоторым критериям: подстроку нельзя расширить и сдвинуть вправо.

2 Описание решения поставленной задачи

2.1 Наивный алгоритм

Самым простым алгоритмом решения поставленной задачи является предложенный организаторами квадратный алгоритм, он же является оптимальным, если матрица $L[i][j]$ помещается в кэш процессора. Так как при вычислении i-ой строки мы используем только $i-1$ строку, то достаточно хранить только 2 последние строки матрицы, уменьшив тем самым расход памяти. Распараллеливание здесь простое - каждому потоку своя пара строк для сравнения.

2.2 Асимптотически оптимальный алгоритм

Асимптотически оптимальный алгоритм в данной задаче - это построить суффиксное дерево транспонированной первой строки и суффиксное дерево транспонированной второй строки, потом объединить эти два дерева (тяжело реализуемая операция). Тогда $L[i][j]$ - это наименьший общий предок i-го и j-го суффикса. Причём для каждого суффикса i нужно перебирать только те суффиксы j, у которых предок на глубине m(минимальная длина совпадения) совпадает с предком i на глубине m. Построить суффиксное дерево можно за $O(N)$ с помощью алгоритма Укконена, где N - длина строки, слить два дерева можно за $O(N+M)$. Дальше мы тратим $O(K)$ операций, где K - число последовательностей в ответе (без учета того, что последовательность нельзя сдвинуть вправо). Получаем линейный алгоритм за $O(N+M+K)$. Увы, несмотря на маленький размер алфавита, у алгоритма Укконена неприлично большая константа и этот алгоритм нельзя распараллелить. Поэтому для достижения нормального времени работы мы использовали следующий эвристический алгоритм:

2.3 Параллельный алгоритм решения

Строим суффиксное дерево меньшей из двух строк. Перебираем все индексы большей строки и спускаемся вниз по дереву на m. Дальше выписываем все суффиксы, которые ниже вершины, до которой мы дошли. В итоге мы для индекса i получили все индексы j, такие что $L[i][j] \geq m$. Причём если такая вершина на глубине m существует, то пройдя по суффиксной ссылке из этой вершины мы попадём в вершину на глубине $m-1$ для индекса $i-1$. Если перебирать индексы i последовательно, начиная с конца, а все такие индексы j хранить в map, то переход от $i+1$ к i примет вид:
1) пройти по суффиксной ссылке из вершины для индекса $i+1$.
2) получить все индексы j, что $L[i][j] \geq m$
3) для всех таких j посмотреть если $L[i+1][j+1] \neq 0$, то $L[i][j] = L[i+1][j+1]-1$, иначе вычислить $L[i][j]$.
4) Добавить в ответ те индексы j, которые удовлетворяют фильтрам задачи (нельзя сдвинуть вправо и расширить).

Этот алгоритм легко можно распараллелить, например, каждому потоку можно дать свой кусок индексов $first...last$ для проверки. Увы, алгоритм всё равно остаётся весьма медленным. Поэтому применяется следующая оптимизация - для всех возможных подстрок длины k вычисляется их положение при спуске с дерева. Тогда первый прыжок делается не на 1, а сразу на k. Естественно, возникает желание взять k побольше, но одновременно не слишком большое, поскольку на это используется 4^k памяти(map использовать нельзя, поскольку он погубит производительность).
Изначально было взято $k = 8$, программа значительно ускорилась. Потом $k = 12$ и несмотря на то, что индексы 4^{12} строк вычислялись моментально, сама программа замедлилась. В дальнейшем стало понятно, что при $k = 8$ потоки постоянно лезут за индексом в кэш и всё работает моментально, а при $k = 12$ они постоянно промахиваются мимо кэша и лезут в оперативную память. Оптимальным значением для $Xeon$-ов оказалось $k = 10$.

2.4 Особенности практической реализации

Решение было написано на языке $C++$. В качестве библиотеки для распараллеливания использовались `pthreads`, поскольку несмотря на свою громоздкость обладают максимальной производительностью. Важными моментами реализации мы считаем:
1) Отказ от векторов и динамических выделений памяти в реализации суффиксного дерева.
2) Не рекурсивная версия алгоритма Укконена.
3) Использование `bfs`, вместо `dfs` для обхода дерева. Рекурсивный `dfs` может упасть из-за нехватки стэка.

2.5 Сравнение времени работы программы при разном количестве используемых процессоров

Для тестирования ускорения использовали команду:

```
time ./run X 16 test_input_3.fa (дальше идут все хромосомы, которые были в архиве)
```

Тесты проводились на системе с 2 x `Xeon E5645`.

Количество потоков	Время (sec)	Ускорение
1	58.19	1.00
2	31.12	1.86
4	20.06	2.90
12	13.19	4.41

Список литературы

[1] *Dan Gusfield* Algorithms on Strings, Trees, and Sequences.

[2] *Богачев К. Ю.* Основы параллельного программирования. М.: Бином, 2003.