АЛГОРИТМ ПОИСКА СОВПАДАЮЩИХ ПОДСТРОК ДВУХ ГЕНОМОВ И ЕГО РАСПАРАЛЛЕЛИВАНИЕ

Е.А.Коваль, О.А. Коваль

Объединенный институт ядерных исследований, Дубна

Введение

В последние годы поиск совпадающих последовательностей ДНК, так называемая (DNA contamination problem), являющихся основным компонентом геномов, превратилось в неотъемлемую часть генной инженерии.

Причины значительного количества совпадений довольно больших участков в геномах, а также высокого разнообразия их состава до конца не изучены. До недавнего времени алгоритмы поиска точно совпадающих подстрок были слишком ресурсозатратны и трудоемки для поиска совпадающих последовательностей больших геномов, таких, как геномы животных и растений, и позволяли находить лишь приближенную схожесть участков геномов (BLAT [1] и др.). Появление новых алгоритмов поиска точно совпадающих последовательностей (в частности Refined repetitive sequence searches utilizing a fast hash function and cross species information retrievals [2], SSAHA [3], ACMES [4]) позволило группе Jeff Reneker-а из Университета Миссури получить выдающиеся результаты, в частности, были обнаружены *ультра консервативные элементы (УКЭ) – достаточно большие участки геномов, имеющиеся в абсолютно идентичном виде в геномах не только многих животных, но и растений* [5]. Полагают, что они играют важную роль в стабилизации и поддержании структуры хромосом, участвуют в «узнавании» и правильном их расхождении во время митоза и мейоза, выступают точками рекомбинации, выполняют функцию защиты теломер и участвуют в регуляции активности генов. Неудивительно, что они привлекают внимание исследователей. Поэтому важной задачей является поиск новых генетических маркеров.

При этом приходится обрабатывать огромные количества информации – сравнивать геномы друг с другом на совпадение подстрок любой длины, большей некоторой, причем размеры геномов млекопитающих достигают порядка гигабайтов символов (1024^3 элементов). При этом разработка эффективных по требованиям к памяти быстрых параллельных алгоритмов выступает на первый план.

1. Описание задачи и простейший алгоритм решения проблемы

Задача: Поиск совпадающих подстрок, длиной более некоторой минимальной, при попарном сравнении «эталонного» генома с набором других геномов.

Опишем задачу на простейшем примере.

Имеется «эталонный» геном и набор сравниваемых с ним геномов:

- Reference (эталонный геном): TTAAGATCAATT,
- первый сравниваемый геном (Test_input1): TTGATCTT,
- второй сравниваемый геном (Test_input2): TTGATCTTAGCAT.

Задача заключается в нахождении в них одинаковых подстрок, например длиной более двух символов, причем если они пересекаются, то мы объединяем их в более длинную подстроку. Принципиальный простейший и очень наглядный алгоритм поис-

ка совпадающих подстрок иллюстрируют представленные ниже рисунки. Заранее мы не знаем, какие подстроки двух геномов совпадают, поэтому алгоритмы поиска заранее известного паттерна в строке, такие, как алгоритмы Кнута–Морриса–Пратта, Бойера–Мура и другие, неприменимы (мы не упомянули алгоритм Рабина–Карпа, т.к. обобщение его идеи послужило базисом нашего решения). Простейший алгоритм «грубой силы» состоит в составлении таблицы совпадения каждого символа эталонного генома с каждым символом сравниваемого генома и выделении наиболее длинных цепочек рядом стоящих символов (указано стрелками на рис. 1). В качестве ответа выводятся положение подстроки (номер начального символа и конечного символа) в эталонном геноме и положение аналогичного участка в сравниваемом геноме.

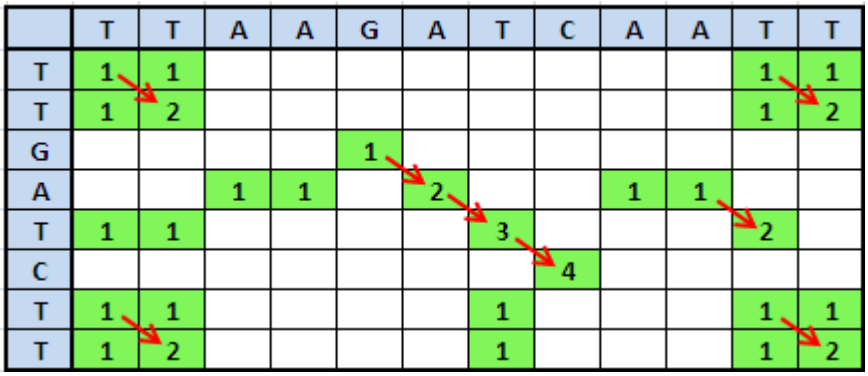

Рис. 1. Простейший алгоритм решения проблемы нахождения совпадающих подстрок. Таблица сравнения reference строки (по горизонтали) и test_input1 (по вертикали)

Из рис.1 можно видеть, что ответом на поставленную задачу являются подстроки (напомним, что ищутся совпадающие подстроки длиной больше либо равные 2):

- test_sequence_1
  1 2 1 2 (TT)
  1 2 7 8 (TT)
  5 8 3 6 (GATC)
  10 11 4 5 (AT)
  11 12 1 2 (TT)
  11 12 7 8 (TT)

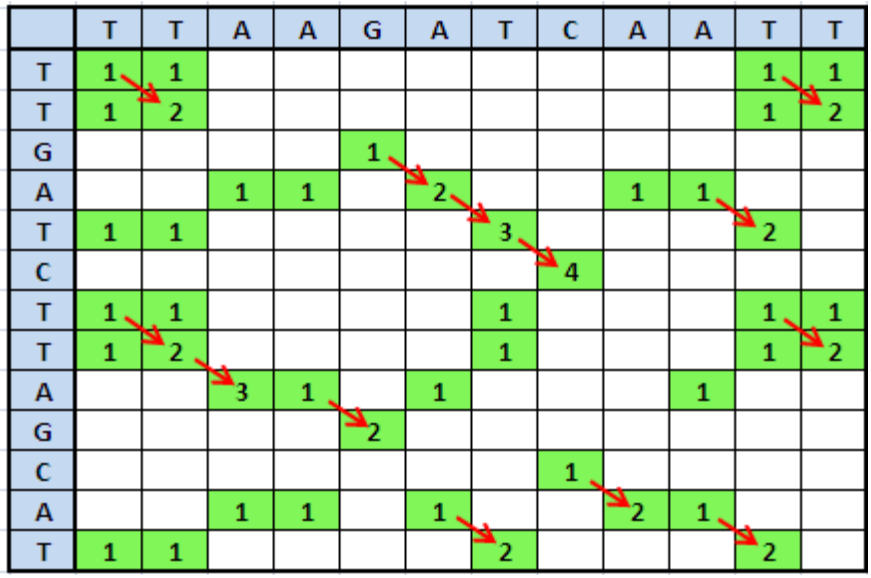

Рис. 2 Таблица сравнения reference строки (по горизонтали) и test_input2 (по вертикали)

Аналогично решением для второго генома из примера, как следует из рис. 2, являются подстроки:

- test_sequence_2
  1 2 1 2 (TT)
  1 3 7 9 (TTA)
  4 5 9 10 (AG)
  5 8 3 6 (GATC)
  6 7 12 13 (AT)
  8 9 11 12 (CA)
  10 11 4 5 (AT)
  10 11 12 13 (AT)
  11 12 1 2 (TT)
  11 12 7 8 (TT)

Задача решалась в рамках конкурса Intel Accelerate 2012 и организаторы любезно предоставили участникам реализацию простейшего алгоритма в виде референсного кода. Таким образом, задача в рамках конкурса ставилась в виде: ускорить и распараллелить референсное решение [6] с полным сохранением условий вывода. (На входе имеются *reference* и набор *input* строк, необходимо найти все совпадающие подстроки длиной больше *minMatchLength* с дополнительным условием фильтрации результатов: максимизирующими тройку (i,j,length), где i – индекс подстроки в *reference* строке, i – индекс подстроки в *input* строке).

1.1. Описание входных данных

Входные данные

В качестве входных данных используются файлы геномов в формате FASTA (.fa), наиболее распространенном формате для хранения и обработке геномов:

```
>reference
GATGAGCATGTGTTGAATCCTCA
>test_input_1
GATGAGCATGTGTTGAATCCTCAGATGAGCATGTGTTGAATCCTCA
>test_input_2
  GTCCTCCAGTTTGTAGCATGTGTATTTATTAATTCTGTCATGTGCAAATGGTTGTCCTGCATATAAAAG
  TATCTGGTCATTTCAGTTTGGTTTGTA
```

***Вывод*:**

```
>test_input_1
1 23 1 23
1 23 24 46
>test_input_2
5 13 15 23
```

Пример вызова программы

```
./run 8 22 refseq.txt input1.txt input2.txt …
```

где 8 – количество потоков, разрешенных создавать программе, 22 – минимальная длина совпадающих подстрок (minMatchLength).

1.2. Алгоритм решения задачи

Последовательный алгоритм

Классическое решение проблемы – построение суффиксных структур, описанных в книге Gusfield D. Algorithms on strings, trees and sequences [7]. Мы не будем их здесь разбирать.

Нашим решением было использование модификации алгоритма Рабина–Карпа, позволяющего достичь тех же времен, что и при построении суффиксных структур. Алгоритм в большей степени ориентирован на **поиск большого количества коротких**

паттернов. Это как раз тот случай, когда имеется много паттернов поиска и необходимо производить поиск сразу всех паттернов.

Идея алгоритма Раббина–Карпа: заменим поиск совпадающих подстрок на поиск совпадений хэш-функций от подстрок, значения которых можно вычислять за линейное время.

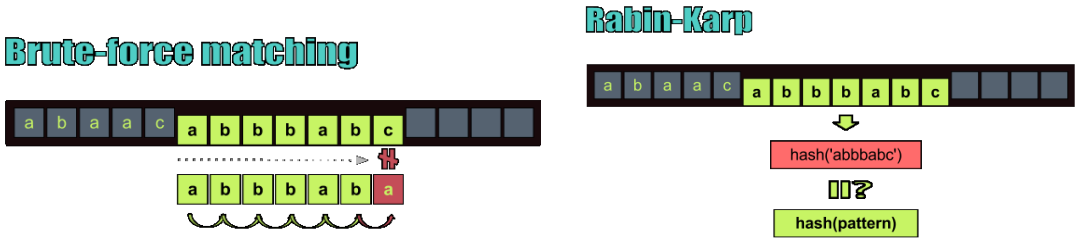

Рис. 3. Идея алгоритма Рабина–Карпа – заменить сравнение подстрок сравнением хэш-функции от подстрок, которую можно вычислять быстро

В качестве хэш-функции была выбрана кольцевая хэш-функция из работы [1], полностью отображающая подстроки на значения хэш-функции, таким образом, *мы избегаем сравнения подстрок между собой*:

$$f(substr) := \sum_{r=1}^{m} f(substr[r]) \cdot 4^r, \qquad (1)$$

$$f(A) := 1,\ f(G) := 2,\ f(T) := 3,\ f(C) := 4.$$

Вычисление хэша следующей подстроки, использующего хэш подстроки, сдвинутой на 1 символ назад, основан на свойствах функции f и действует как «скользящее окно» – достаточно вычесть значение первого элемента (слева), поделить оставшееся на 4 и прибавить значение нового элемента (справа), умноженное на 4^m, т.е. окно как бы двигается как целое слева направо и вычисление хэш значения следующей подстроки из хэш значения текущей подстроки вычисляется за O(1).

Выбор $f(A) := 1$ не нулем, а единицей, сделан для различения подстрок A и AAAAAA.

Таким образом, поиск сводится к построению хэш-таблицы от *input* строк с парами – ключ (значение хэш-функции от подстроки) и положение её первого элемента, – и быстрым вычислением благодаря последовательному сдвигу хэш-функции от подстрок *reference* строки («эталонного» генома) с поиском вычисленного значения в хэш-таблице. Нахождение значения в хэш-таблице означает обнаружение совпадающей подстроки длин *m*. Найденные совпадения записываются в хэш-таблицу результатов и фильтруются после прохождения основного этапа для отбора как возможно более длинных совпадающих подстрок.

Пример вычисления хэш-функции от первого сравниваемого генома *test_input_1* (“TTGATCTT” → “22312422”) для *minMatchlength*=3:

“223” → $2*4^1+2*4^2+3*4^3 = 8 + 32 + 192 = 232$

“231” → $(232 – 2*4^1)/4 + 1*4^3 = 56 + 64 = 120$ и т.д.

Keys, f(substr)	I, index of substr in ref.string
120	2
176	6
...	...
192	5
232	1

Рис. 4. Хэш-таблица подстрок первого сравниваемого генома

Пробег по значениям хэш-функции на подстроках reference строки с поиском каждой в хэш-таблице находит все совпадающие подстроки длиной *m* равной 31, если minMatchlength больше 31, и самому minMatchlength, если меньше. Ограничения на максимальное значение *m* в виде 31 определяется из пределов значений целого типа unsigned long long, в которых мы можем хранить значения хэш-функции.

Использование хэш-таблиц предоставляет следующие преимущества: хранение только заполненных данных, быстрота обращения к ячейке (по сравнению с деревьями).

Результатом работы алгоритма будет хэш-таблица со значениями положений совпадающих подстрок длиной *m* в исходных строках.

2. Параллелизация, улучшения алгоритма и результаты

Если ***Длина максимального иннпута > Длина reference строки,*** то меняем ролями *reference* и *input* строки – хэш-таблицу строим от *reference* строки, а пробегаемся по иннпутам.

Это более медленная операция, т.е. сводя все хэш-таблицы от разных иннпутов в одну с сортировкой по хэшу, мы существенно экономим время поиска.

Параллелизация по стратегии «Разделяй и властвуй» "***Divide and conqueror***". Каждому потоку отдается часть reference строки в которой он запускает последовательный алгоритм. Достаточно поделить поиск подстрок в *reference* (или *input*) строке между потоками, они абсолютно независимы после заполнения хэш-таблицы инпутов (*reference* строки). Пример для случая двух потоков показан на рис. 5.

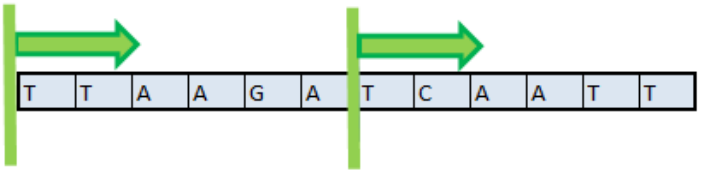

Рис. 5. Стратегия «Разделяй и властвуй» для случая двух потоков

3. Результаты

В результате проведенных тестов замеры времени работы программ показали (табл. 1), что в среднем *ускорение работы алгоритма* с реализацией хэш-таблицы в виде STL *map* (естественно, не лучшем) составляет 19 *раз*.

Таблица 1. Ускорение алгоритма по сравнению с алгоритмом «грубой силы»

Тест	Количество потоков	Алгоритм «грубой силы»	Хэш алгоритм	Ускорение
refseq_3 (5 kb) vs input_3 (200 Kb)	1	22.3 sec	1.197 sec	в 18.8 раз

4. Масштабируемость

Использование реализации хэш-таблицы в виде *map* структуры библиотеки STL не позволило добиться максимальной масштабируемости в силу многопотоковой небезопасности структуры. Для улучшения масшатабируемости рекомендуется использовать хэш-таблицы библиотеки Intel Threading Building Blocks (TBB) или Google SparseHash, DenseHash.

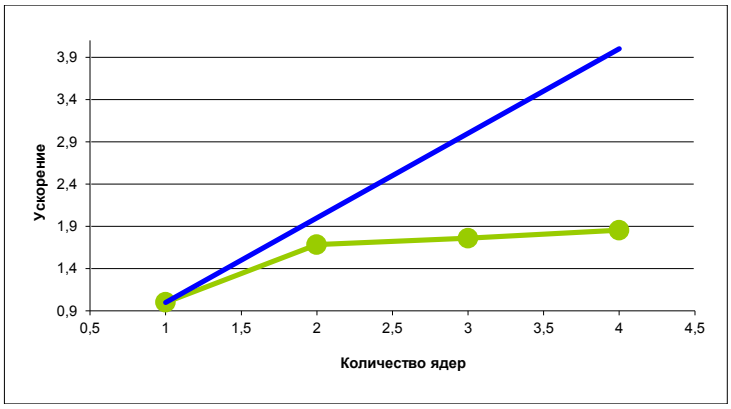

Рис. 6. Core i7 (total : 4 cores) with 8Gb of ram and a 500Go Hard Drive

5. Заключение

Преимуществами алгоритма являются простота реализации и хорошая эффективность, особенно для экстремальных задач – длина reference или одной из input строк много больше, чем длины остальных строк. *Сравнение большого количества коротких геномов с большим «эталонным» – наилучший случай для оптимальной скорости работы алгоритма*. Существует большое количество эффективных и по скорости, и по памяти хэш-таблиц для эффективной реализации в коде. Также представленный алгоритм прост в распараллеливании.

Литература

1. Altschul S., Gish W., Miller W., Myers E., Lipman D. Basic local alignment search tool // Journal of Molecular Biology. 1990. 215. P. 403-410.
2. Reneker J., Shyu C.-R. Refined repetitive sequence searches utilizing a fast hash function and cross species information retrievals // BMC Bioinformatics. 2005. 6: 111 – [http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1131890/].
3. Ning Z., Cox A., Mullikin J. SSAHA – A Fast Search Method for Large DNA Databases // Genome Research. 2001. 11: P. 1725-1729.
4. Ning Z., Cox A., Mullikin J. SSAHA – A Fast Search Method for Large DNA Databases // Genome Research. 2001. 11: P. 1725-1729 – [http://www.biomedcentral.com/pubmed/15215469].
5. Reneker J., Lyons E. et al. Long identical multispecies elements in plant and animal genomes // Proc Natl Acad Sci USA. 2012. May 8;109(19) – [http://www.ncbi.nlm.nih.gov/pubmed/22496592].
6. Accelerate 2012 Parallel Programming Contest – [http://software.intel.com/ru-ru/articles/contest-accelerate-2012-problem/].
7. Gusfield D. Algorithms on Strings, Trees and Sequences. Computer Science and Computational Biology. Cambridge, UK: Cambridge University Press, 1997.