Массовый поиск подстрок с использованием CUDA

Содержание

1. Введение

Задача поиска подстрок наряду с сортировкой и поиском – одна из самых распространенных задач информатики, используемая в различных областях computer science. В связи с этим существует огромное количество методов и алгоритмов ее решения. В настоящее время наибольшее распространение среди универсальных алгоритмов получили вариации на тему классических методов Бойера-Мура (Boyer-Moore) [1], КМП – алгоритма Кнута-Морриса-Пратта (Knuth-Morris-Pratt) [2] и Ахо-Корасик (Aho-Corasick) [3]. Эти методы дают хорошие результаты при поиске одиночной подстроки в строке данных.

Кроме того, существует ряд алгоритмов учитывающий априорные знания об особенностях входных данных и, за счет этого, имеющие на определенном классе задач лучшие характеристики по сравнению с универсальными методами. Например, существуют алгоритмы для быстрого поиска английских слов в тексте, для поиска на задачах с очень большим или очень маленьким алфавитом, для поиска коротких подстрок и пр.

Отдельной областью применения алгоритмов поиска подстрок является массовый поиск. При этом в одном и том же участке данных производится поиск множества подстрок. Очевидное решение этой задачи – для каждой из подстрок применять один из методов, описанных выше. Но также очевидно, что такой подход ведет к множеству избыточных проверок. В настоящее время используют либо алгоритмы поиска подстроки адаптированные для поиска нескольких подстрок (как правило на базе алгоритма Ахо-Корасик) либо алгоритмы специально разработанные для массового поиска. Один из самых распространенных и красивых решений этой задачи - метод Рабина-Карпа [4]. Наиболее существенной особенностью этого подхода является его ориентированность на поиск подстрок одинаковой длинны. Существуют вариации для массового поиска подстрок произвольной длины, но они, к сожалению не так эффективны, как оригинальный алгоритм.

Кроме того, абсолютное большинство алгоритмов поиска подстрок, за исключением, быть может алгоритма shift-or [5] не учитывают особенности архитектуры на которой они будут исполняться, что ведет к неэффективному использованию возможностей кэширования обращений к памяти, неоптимальному использованию возможностей конвейеризации, предсказания переходов, и других особенностей работы вычислительной системы на которой планируется производить поиск.

Задача поиска подстрок наряду с сортировкой и поиском - одна из самых распространенных задач информатики, используемая в различных областях computer science. В связи с этим существует огромное количество методов и алгоритмов ее решения. В настоящее время наибольшее распространение среди универсальных алгоритмов получили вариации на тему классических методов Бойера-Мура (Boyer-Moore) [1], КМП - алгоритма Кнута-Морриса-Пратта (Knuth-Morris-Pratt) [2] и Ахо-Корасик (Aho-Corasick) [3]. Эти методы дают хорошие результаты при поиске одиночной подстроки в строке данных.

Кроме того, существует ряд алгоритмов учитывающий априорные знания об особенностях входных данных и, за счет этого, имеющие на определенном классе задач лучшие характеристики по сравнению с универсальными методами. Например, существуют алгоритмы для быстрого поиска английских слов в тексте, для поиска на задачах с очень большим или очень маленьким алфавитом, для поиска коротких подстрок и пр.

Отдельной областью применения алгоритмов поиска подстрок является массовый поиск. При этом в

одном и том же участке данных производится поиск множества подстрок. Очевидное решение этой задачи - для каждой из подстрок применять один из методов, описанных выше. Но также очевидно, что такой подход ведет к множеству избыточных проверок. В настоящее время используют либо алгоритмы поиска подстроки адаптированные для поиска нескольких подстрок (как правило на базе алгоритма Ахо-Корасик) либо алгоритмы специально разработанные для массового поиска. Один из самых распространенных и красивых решений этой задачи - метод Рабина-Карпа [4]. Наиболее существенной особенностью этого подхода является его ориентированность на поиск подстрок одинаковой длинны. Существуют вариации для массового поиска подстрок произвольной длины, но они, к сожалению не так эффективны, как оригинальный алгоритм.

Кроме того, абсолютное большинство алгоритмов поиска подстрок, за исключением, быть может алгоритма shift-or [5] не учитывают особенности архитектуры на которой они будут исполняться, что ведет к неэффективному использованию возможностей кэширования обращений к памяти, неоптимальному использованию возможностей конвейеризации, предсказания переходов, и других особенностей работы вычислительной системы на которой планируется производить поиск.

2. Постановка задачи

Дано:

- алфавит данных Σ;
- буфер поиска H длиной $|H| = h$;
- множество подстрок для поиска $\{N_i\}$ количеством $|N| = n$.

Необходимо произвести поиск (установить факт наличия и, возможно, местоположение) множества подстрок различной длины N в буфере данных H. Поиск производится на полном восьмибитном алфавите, т.е. $|\Sigma| = 256$. Известно также, что $\max\limits_{i=1}^{n} |N_i| < h$.

3. Предлагаемый метод

Для массового поиска подстрок предлагается реализовать следующий простой метод.

3.1. Предварительная обработка

1. Для каждого символа входного алфавита $\sigma_i \in \Sigma$, создается набор из всех вхождений этого символа в множество N подстрок поиска в виде пар $(n, k), n \in N$ состоящих из подстроки n и индекса k символа σ_i внутри n.
2. Формируется рабочая матрица $R = \{r\}$ размером $|N| \times |H|$. Каждая строка i рабочей матрицы заполняется значением равным количеству символов в подстроке N_i.

3.2. Основная итерация

Для каждого символа входного буфера $c_i \in H, i = 0..h-1$ выполняются следующие действия

1. Находятся все пары (n, k) соответствующие символу c_i.
2. Для каждой найденной пары (n, k) производится декремент элемента рабочей матрицы $r_{n,i-k}$

3.3. Интерпретация результатов

После обработки всех символов входного буфера, каждый нулевой элемент рабочей матрицы $r_{i,j} = 0$ соответствует подстроке N_i найденной во входном буфере начиная с позиции j.

3.4. Модификации

Возможно обратная обработка рабочей матрицы, то-есть ее инициализация нулями и инкремент в основной итерации. В этом случае, подстрока считается найденной в случае совпадения значения элемента рабочей матрицы с длиной соответствующей подстроки.

Для поиска только факта наличия подстрок или количества вхождений подстрок, метод поиска может быть изменен очевидным образом.

4. Требования к реализации

4.1. Входные данные

- входные данные алгоритма представляют собой строки, состоящие из случайных последовательностей байт;
- входные данные алгоритма формируются автоматически на базе параметров входных данных;
- должна быть предусмотрена возможность сохранения входных данных алгоритма в файл, для проверки корректности работы;
- параметры входных данных алгоритма задаются при старте программы;
- параметры входных данных алгоритма включают в себя:
  - длину буфера h для поиска подстрок;
  - количество подстрок $|N|$;
  - минимальную длину подстроки $\min\limits_{i=1}^{n} |N_i|$;
  - максимальную длину подстроки $\max\limits_{i=1}^{n} |N_i|$;
  - вариант поиска – поиск факта присутствия подстроки в строке/поиск всех позиций подстроки в строке;

4.2. Выходные данные

- время работы алгоритма с использованием GPU;
- время работы алгоритма без использования GPU;
- отчет о совпадении/отличии результатов работы алгоритма с и без использования GPU;
- результаты поиска - номера (и, возможно, позиции начала) найденных подстрок.

4.3. Реализация

Для увеличения производительности на GPU можно использовать атомарные операции и кэширование доступа к памяти.

5. Ожидаемый результат

1. Знакомство с методами поиска подстрок.
2. Получение/углубление навыков написания и отладки программ с использованием CUDA.
3. Использование текстур, атомарных операций, а также, возможно асинхронных операций передачи данных между основной памятью и памятью устройства CUDA.

Список литературы

[1] Boyer R. S., Moore J. S. A Fast String Searching Algorithm // Communications of the ACM. 1977. T. 20, № 10. c. 762.

[2] Fast Pattern Matching in Strings / D. E. Knuth, , J. H. Morris [и др.] // SICOMP: SIAM Journal on Computing. 1977. T. 6.

[3] Aho A. V., Corasick M. J. Efficient String Matching: An Aid to Bibliographic Search // CACM: Communications of the ACM. 1975. T. 18.

[4] Introduction to Algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest [и др.]. Cambridge, MA: MIT Press, 2001.

[5] URL: http://www.cs.uku.fi/ fredriks/pub/papers/jda09.pdf.