Санкт-Петербургский Государственный Университет
Математико-механический факультет
Кафедра системного программирования

Аппаратное ускорение задачи выравнивания строк на языке HaSCoL

Курсовая работа студента 345 группы
Найданова Дмитрия Геннадьевича

Научный руководитель Медведев О. В.

Санкт-Петербург, 2011

Содержание

1 Введение

Целью данной курсовой работы является создание аппаратно-программного комплекса, который позволит решать задачу множественного выравнивания строк.

В биоинформатике, выравнивание строк используется для определения функциональных, структурных или эволюционных связей между цепочками ДНК, РНК или протеинов. Строки размещаются одна под другой, в них вставляются разрывы. Разрывы обозначаются знаком "-". Одновременно могут выравниваться несколько строк, но эту задачу можно решить приближенно при помощи попарного выравнивания строк.

Различают глобальное и локальное выравнивание. Глобальное выравнивание применяется к «похожим» последовательностям приблизительно одинаковой длины и наглядно показывает разницу между этими последовательностями. Строки выравниваются так, чтобы была максимальна целевая функция $\sum_{i=0}^{length} weight(a_i, b_i)$ (для двух строк), где length — длина строки с разрывами, weight — заданная функция, которая сопоставляет значение двум символам.

Локальное выравнивание обычно применяется для отличающихся друг от друга последовательностей, которые предположительно могут содержат одинаковые участки. Фактически, это поиск двух подстрок с максимальным весом при глобальном выравнивании этих подстрок целиком.
Например,
локальное выравнивание

```
FTFTALILL-AVAV
--FTAL-LLAAV--
```

глобальное выравнивание

```
FTFTALILLAVAV
F--TAL-LLA-AV
```

В качестве программной части используется MAFFT[1].

Для реализации аппаратной части выбран язык HaSCoL[6]. HaSCoL (Hardware - Software Codesign Language) — высокоуровневый язык описания аппаратуры, который разрабатывается на кафедре системного программирования. Язык поддерживает надежную и ненадежную доставку сообщений, неявную конвейеризацию обработчиков. Также язык поддерживает большое число конструкций обычных языков высокого уровня (циклы, массивы и т.д.). Код на хасколе транслируется в VHDL. Вместе с транслятором поставляется стандартный набор типов и операций над ними (bincompl). Также в коде на хасколе возможно использование внешних модулей на VHDL.

Взаимодействие элементов архитектуры в хасколе описывается в терминах передачи и обработки сообщений. Структурным элементом архитектуры является процесс, у которого могут быть входы и выходы. На входы процесса могут быть поданы данные (в таком случае говорят, что процесс получил сообщение), на выходах данные могут появиться сами как результат внутренней деятельности процесса. Сам процесс, помимо входов и выходов, обладает внутренними данными, а также набором обработчиков, каждый из которых может запускаться при выполнении определенного набора условий. Одним из видов условий является наличие входного сообщения. Будучи запущен, обработчик может посылать сообщения через выходы процесса или модифицировать собственные данные процесса.

Язык пока что не имеет большой популярности и у разработчиков мало отзывов от пользователей, поэтому дополнительной целью данной работы — придумать какие-нибудь улучшения, которые помогут сделать язык более удобным.

Также использовался расширенный препроцессор для C++, адаптированный для языка HaSCoL[3].

2 Обзор существующих разработок

Аппаратное ускорение биологических алгоритмов производилось и раньше. Например, в работе [7] было произведено сравнение VHDL и MitrionC как инструментов для программирования FPGA и было ускорено приложение FASTA.

На FPGA была перенесена функция FLOCAL_ALIGN (оптимизированный алгоритм Смита-Вотермана), которая выполняется 98.6% времени работы программы.

Были использованы Xilinx Virtex-II Pro 50 и Virtex-4 LX160, для хранения входных строк используется внутренняя память FPGA, промежуточные данные хранятся во внешеней памяти QDR II SRAM.

В статье была предложена интересная идея насчет схемы прошивки для FPGA: нужно разбить задачу выравнивания двух строк на задачи по выравниванию второй строки с каждым символом первой, учитывая результаты, полученные при выравнивании с предыдущим символом, для каждой такой задачи создать небольшое устройство, объединить все устройства в конвейер. Подобная идея используется и в моей работе.

Было достигнуто ускорение от 10x до 100x по сравнению с исходной программой, запущенной на компьютере с процессором Opteron 2.2GHz, наилучшие результаты были показаны для самых длинных строк.

3 Реализация

Архитектура всей системы Только небольшая часть всей системы может быть помещена на FPGA. В программной части (MAFFT) 80% времени выполнения занимает попарное выравнивание строк, поэтому эта процедура — идеальный кандидат для переноса на FPGA.

В MAFFT используются несколько алгоритмов, но все они очень похожи, поэтому было решено сначала реализовать алгоритм Смита-Вотермана.

Обзор алгоритма Алгоритм Смита-Вотермана предназначен для выравнивания последовательностей нуклеотидов с целью определения их общих частей. Это алгоритм локального выравнивания, т.е. он рассматривает не обе строки целиком, а все их подстроки. Такие алгоритмы нужны для выравнивания сильно различающихся строк.

Смит-Вотерман — алгоритм динамического программирования; вначале строится матрица H следующим образом: $H(i,0) = 0, 1 \leq i \leq n; H(0,j) = 0, 1 \leq j \leq m$, где m, n - длины строк; затем матрица заполняется:

$$H(i,j) = max \begin{cases} H(i-1,j-1) + w(a_i, b_i) \\ H(i,j-1) + w(-, b_j) \\ H(i-1,j) + w(a_i, -) \end{cases}$$

где w - вес для совпадения/несовпадения или разрыва; a_i, b_j - символы строк.

Чтобы получить оптимальное выравнивание, нужно начать с наибольшего значения в матрице и идти назад в $(i-1,j-1), (i-1,j), (i,j-1)$ в зависимости от того, какое направление было использовано для построения матрицы.

Алгоритм был придуман Смитом и Вотерманом в 1981 году, более подробно о нем можно прочитать в [4]

Пример

Строка 1: $ACACACTA$

Строка 2: $AGCACACA$

$w(match) = 2; w(mismatch) = w(a,-) = a(-,b) = -1$

Матрица H будет (см. 1)

Наибольшее значение в ячейке (8,8), из нее нужно идти в (7,7), потом (7,6), (6,5), (5,4), (4,3), (3,2), (2,1), (1,1), и (0,0).

Выравнивание

```
A-CACACTA
AGCACAC-A
```

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & 3 & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & 5 & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & 7 & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & 9 & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & 11 & 10 & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & 12 \end{pmatrix}$$

Рис. 1: Матрица

Архитектура устройства В реализации использованы возможности FPGA по распараллеливанию вычислений: устройство состоит из большого числа “вычислителей“, каждый из которых получает свой символ первой строки и сравнивает его с каждым символом второй строки. Каждый такой “вычислитель“ — хаскольный процесс с обработчиками для символов первой и второй строки. “Вычислители“ соединены в конвейер. Таким образом, верхние строки матрицы заполняются одновременно с первыми столбцами (обрабатываемые ячейки матрицы всегда располагаются по диагонали), поэтому время работы должно быть $O(2 * length(a) + length(b))$.

	A	C	A	T	...	T
A	4	5	9	10	...	100
C	5	9	10	11	...	
C	9	10	14	15		
A	10	11	12			
...	...	...				
C	122					

У устройства три входа (для первой строки, для второй строки и для обозначения конца ввода) и один выход для направлений. Каждые два такта на вход устройства можно подавать один символ первой строки и каждые пять тактов один символ второй строки.

Устройство общается только с внутренней памятью FPGA, что ограничивает максимальную длину выравнимаемых строк. Т.к. в хаскоде нет поддержки работы с памятью, то используется внешний модуль на VHDL.

Тестирование устройства Для тестирования устройства была написана оболочка на SystemC. В качестве симулятора использовалась программа ModelSim. В ходе тестирования были найдены пары строк, на которых ответы, выдаваемые устройством, хуже ответов эталонной реализации.

Производительность Полученный при трансляции код на VHDL был синтезирован при помощи утилиты xst от Xilinx для устройства Virtex5 xc5vlx50t-1, удалось получить частоту в 200МГц, что, без учета накладных расходов на передачу данных между PC и FPGA, дает возможность обрабатывать 40 млн символов в секунду. Эталонная реализация обрабатывает 20 млн. символов за 29 секунд на компьютере с процессором Pentium T2370 1.73 GHz 1Mb L2 cache.

Устройство из 8 вычислителей заняло 56% look up таблиц и 8% регистров.

4 Заключение

Был проведен обзор существующих решений задачи. На основе обзора был реализован прототип аппаратной части комплекса, который способен выравнивать строки равной фиксированной длины. Было проведено тестирования с использованием оболочки на SystemC и симулятора ModelSim. Для получения информации о производительности устройства была использована утилита xst от Xilinx.

В дальнейшем планируется доработать устройство и соединить его с программной частью.

Исходный код прошивки и тестовой оболочки можно найти на
http://code.google.com/p/hascolplayground/source/browse/#svn%2Ftrunk%2Fndmitry91%2Ffinal

Список литературы

[1] Сайт MAFFT: http://mafft.cbrc.jp/alignment/software/

[2] Сайт FASTA: fasta.bioch.virginia.edu

[3] Препроцессор http://www.kouzdra.ru/software.gpc.html

[4] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences http://gel.ym.edu.tw/~chc/AB_papers/03.pdf

[5] Короткое описание алгоритма Смита-Вотермана http://en.wikipedia.org/wiki/Smith-Waterman_algorithm

[6] Сайт проекта CoolKit, транслятора для HaSCoL http://oops.math.spbu.ru/projects/coolkit/wiki/WikiStart

[7] Performance Evaluation of FPGA-Based Biological Applications, Olaf Storaasli and Weikuan Yu, ORNL, Dave Strenski, Cray, Jim Maltby, Mitrionics http://ft.ornl.gov/~olaf/pubs/CUG07Olaf17M07.pdf

[8] Обзор некоторых высокоуровневых языков для программирования FPGA http://fpga.parallel.ru/lang.html