Интернет-сервис для построения множественного выравнивания последовательностей на многопроцессорных системах, созданный на основе data-flow модификации алгоритма MUSCLE *

А. Н. Сальников

В статье обсуждается модификация алгоритма MUSCLE, которая позволяет строить множественное выравнивание нуклеотидных и аминокислотных последовательностей на многопроцессорной технике. В оригинальный алгоритм MUSCLE внесены изменения, которые реализуют концепцию программы, управляемой потоком данных. В итоге получается MPI реализация, осуществляющая динамическое распределение потоков данных с помощью системы PARUS. Алгоритм протестирован на выравнивании всех LTR5 объектов в геноме человека. Для облегчения трудностей при использовании многопроцессорной техники конечному пользователю предоставляется Веб-интерфейс, через который он может ставить задачи по выравниванию последовательностей в очередь на многопроцессорной системе.

1. Введение

Исследование структуры аминокислотных и нуклеотидных последовательностей чрезвычайно важно для молекулярной биологии и биоинженерии. Одним из достаточно важных инструментов для проведения исследований является инструмент множественных выравниваний. Он предназначен для систематизации ДНК, РНК и белковых фрагментов и для поиска участков схожести, которые могут соответствовать функциональным, структурным или эволюционным отношениям между данными фрагментами.

Выравниваемые последовательности обычно представляются как строки некоторой матрицы. В изначальные последовательности между аминокислотными или нуклеотидными остатками, которые представлены как буквы в алфавите, вставляются специальные символы “гэпы” таким образом, чтобы идентичные или похожие символы выстроились в колонку.

На сегодняшний день существует несколько довольно популярных алгоритмов и реализующих их программ в том числе: ClustalW [3], MUSCLE [5], DIALIGN-TX [9]. Для алгоритмов ClustalW и DIALIGN-TX существуют их параллельные реализации ClustalW-MPI [4] и Dialign P [10] соответственно. Однако Алгоритм ClustalW несколько устарел по отношению к более современным алгоритмам, а Dialign P не позволяет выравнивать большое число последовательностей в связи с требовательностью к ресурсам памяти. В связи с этим было принято решение о распараллеливании более современного алгоритма MUSCLE.

Молекулярному биологу иногда приходится выравнивать либо большое число коротких последовательностей, либо небольшое число длинных. Для построения таких выравниваний необходимы существенные вычислительные ресурсы, которые могут быть обеспечены современными многопроцессорными вычислительными системами коллективного использования. Часто молекулярные биологи не являются специалистами в областях, связанных с использованием многопроцессорных систем, поэтому важно создать понятный для биолога интерфейс (например, Веб-интерфейс), который позволит ставить задачи по выравниванию последовательностей на одной из нескольких зарегестрированных для данного интерфейса многопроцессорных систем.

*Работа проводится при поддержке грантов: РФФИ 08-07-00445 и президента Российской Федерации МК-1606.2008.9

2. Алгоритмы выравнивания последовательностей

2.1. Парное выравнивание

Для понимания вычислительных процессов, связанных с построением множественного выравнивания, обратимся сперва к парному выравниванию. Определим две последовательности символов следующим образом. Последовательности $S_1 = s_{1,1}s_{1,2}...s_{1,L_1}$ и $S_2 = s_{2,1}s_{2,2}...s_{2,L_2}$ алфавита $A = \{\alpha_1, \alpha_2, ..., \alpha_n\}$. Матрица $R = ((r_{i,j}))$ размерности $2 \times L$ называется парным выравниванием последовательностей S_1 и S_2, если каждая позиция $r_{i,j}$ матрицы содержит либо символ алфавита A, либо символ "-", который называется *indel* символ (комбинация слов "insertion" (вставка) и "deletion" (удаление)) [5]). Матрица R должна удовлетворять следующим условиям:

1. строка $R_1 = r_{1,1}r_{1,2}...r_{1,L}$ матрицы R получена из строки S_1, где в некоторых позициях вставлены символы indel, следовательно $L_1 \leq L$.
2. строка $R_2 = r_{2,1}r_{2,2}...r_{2,L}$ матрицы R получена из строки S_2, где в некоторых позициях вставлены символы indel, следовательно $L_2 \leq L$.
3. В матрице R отсутствуют столбцы, состоящие целиком из символов indel.

Среди всех возможных парных выравниваний ищется выравнивание, оптимальное относительно некоторой оценочной функции. Оценочная функция обычно отражает биологический смысл выравнивания. Рассмотрим задачу реконструкции эволюции аминокислотной последовательности белков. Белок является линейным гетерополимером, состоящим из мономеров 20 различных типов — аминокислотных остатков. Пусть каждый тип мономеров задаётся одним из символов определённого алфавита, тогда любой химической формуле белка можно однозначно сопоставить последовательность символов алфавита A. Предположим, что выравниваемые последовательности имеют общую последовательность предка. В процессе эволюции наблюдаемые последовательности эволюционировали от последовательности предка посредством следующих операций (мутаций):

1. замена символа,
2. удаление символа,
3. вставка символа.

Теоретически, биологически корректное выравнивание показывает эволюцию 2-х различных последовательностей, имеющих общего предка. Мы можем попытаться воссоздать корректное выравнивание, если большая часть символов в последовательностях не мутировала. Однако, на практике это можно сделать чрезвычайно редко, поскольку эволюция длится сотни миллионов лет.

Стоит отметить, что не все подстановки символов равновероятны: аминокислотные остатки подразделяются по степени схожести в зависимости от своих физико-химических свойств (заряду, гидрофильности/гидрофобности, размеру и т.п.), вследствие этого замена схожих остатков одного на другой происходит чаще.

Таким образом, естественной видится стратегия, когда максимизируется число схожих символов в одинаковых позициях строк выравнивания при минимизации штрафов за длину вставленных гэпов в строки (подряд идущие символы indel). В статье [2] показаны принципы формирования оценочных функций для выравниваний. Подробно рассмотрим одну из возможных оценочных функций:

$$score(R_1, R_2) = \sum_{i=1}^{L} substitute(r_{1,i}, r_{2,i}) - \sum_{j=1}^{G} gap\_penalty(g_j),$$

где g_j – один из гэпов. (1)

Функция $substitute(r_{1,i}, r_{2,i})$ определяет степень приемлемости того факта, что один символ находится напротив другого в соответствующих позициях выравнивания. Если $r_{1,i}$ или $r_{2,i}$ являются символом *indel*, то функция будет вычислена с 0 значением. Пусть P_a — вероятность встретить аминокислотный остаток a в соответствующей последовательности, $P_{a,b}$ — вероятность замены a на b в процессе эволюции. В этом случае функция подстановок может быть вычислена следующим образом: $substitute(a, b) = \ln(\frac{P_{a,b}}{P_a P_b})$, где a и b символы алфавита A.

Функция штрафов $gap\_penalty(g_j)$ задаёт размер штрафа за вставленный гэп. Эта функция вычисляется для всех гэпов в обоих строках выравнивания (G — множество всех гэпов выравнивания, а g_j — один из них) следующим образом:

$$gap\_penalty(g) = cost_{begin} + length(g) \cdot cost_{extension}.$$

Здесь: $cost_{begin}$ — штраф за инициацию гэпа, $cost_{extension}$ — штраф за удлинение на один символ. Штраф за удлинение обычно меньше чем штраф за инициализацию, что исключает возможность появления большого количества коротких гэпов.

Итак, проблема поиска выравнивания сводится к поиску такой матрицы R для заданных строк S_1 и S_2, на которой достигается максимум оценочной функции. Оптимальное парное выравнивание ищется с помощью метода динамического программирования (Алгоритм Нидельмана – Вунша [1]).

2.2. Множественное выравнивание

Множественное выравнивание (выравнивание более чем 2-х последовательностей) определяется по аналогии с парным выравниванием как матрица $R = ((r_{i,j}))$ размерности $M \times L$, где M — число выравниваемых последовательностей.

Метод динамического программирования позволяет найти оптимальное парное выравнивание за приемлемое время. Но для множественного выравнивания многомерный вариант динамического программирования имеет экспоненциальную временную сложность относительно числа выравниваемых последовательностей. Поэтому обычно используют многочисленные эвристические алгоритмы. Эвристические алгоритмы не гарантируют точного решения относительно некоторой оценочной функции, но имеют значительно меньшую сложность и дают приемлемое решение с точки зрения биологии.

Далее будет рассмотрен алгоритм, описанный в статье [5]. Он реализован в виде одноимённой программы, доступной с сайта http://www.drive5.com/muscle. Этот алгоритм в данной работе был выбран как основа для распараллеливания.

2.3. Алгоритм MUSCLE

Алгоритм MUSCLE разделён на несколько стадий.

На первой стадии алгоритм вычисляет степень схожести между всеми выравниваемыми последовательностями и помещает их в матрицу. Далее последовательности кластеризуются по их схожести и в дальнейшем строится бинарное кластерное дерево. Для построения дерева используется алгоритм UPGMA [6].

Степень схожести между отдельными последовательностями вычисляется следующим методом. Все последовательности делятся на множество фрагментов длинны k (к-мер). Для каждой последовательности подсчитывается число различных k-меров, которые её составляют. Степень схожести определяется по числу совпадающих к-меров в различных последовательностях и определяется по следующей формуле:

$$similarity(S_i, S_j) = \sum_{m=1}^{M} \frac{min(frequency(S_i, \tau_m), frequency(S_j, \tau_m))}{min(length(S_i), length(S_j)) - k + 1}.$$

Здесь τ_m один из различных k-меров, присутствующих в последовательности; $frequency(S_i, \tau_m)$ — число раз, которое τ_m встретился в последовательности S_i; M — общее число различных k-меров, присутствующих в последовательности.

После вычисления схожести строятся выравнивания пар листьев, непосредственно имеющих общего предка в дереве. Все такие выравнивания строятся методом динамического программирования, где в качестве оценочной функции используется (1). Далее для всех внутренних вершин дерева, от листьев к корню, строятся промежуточные выравнивания (выравнивания соответствующих подмножеств последовательностей). Они строятся по аналогии с парным выравниванием, то есть как пара последовательностей, но с отличающейся оценочной функцией. Для промежуточных выравниваний, приписанных к какой-то одной ветке дерева, запрещается сдвигать входящие в него последовательности друг относительно друга. То есть гэпы вставляются сразу в одну и туже позицию сразу для всех строк входящих в промежуточное выравнивание. Оценочная функция для промежуточных выравниваний R_x и R_y вычисляется по следующей модификации формулы (1):

$$score(R_x, R_y) = \sum_{i=1}^{L} \sum_{\alpha} \sum_{\beta} f(\alpha, R_{x,i}) f(\beta, R_{y,i}) substitute(\alpha, \beta) - \sum_{j=1}^{G} gap\_penalty(g_j),$$

$$\alpha \in column\_symbols(R_{x,i}),$$
$$\beta \in column\_symbols(R_{y,i}).$$

Функция $column\_symbols(R_{x,i})$ вычисляет множество символов, которые встречаются в столбце i промежуточного выравнивания x. Функция $f(\alpha, R_{x,i})$ определяет частоту встречаемости символа α в столбце i промежуточного выравнивания x.

На второй стадии кластерное дерево улучшается на основе более точного вычисления степени сходства между последовательностями. Для оценки используется расстояние Кимуры [5]. После этого выравнивание перестраивается схожим с первой стадией способом. К счастью, этот процесс не затрагивает все последовательности. Необходимо построить выравнивания только для тех промежуточных выравниваний, которые соответствуют изменённым узлам дерева. Вторая стадия может быть повторена несколько раз.

3. Параллельный data-flow алгоритм построения множественного выравнивания

В исходный алгоритм MUSCLE были внесены некоторые изменения. Это было сделано с целью получения параллельной реализации, пригодной для использования на многопроцессорных системах, в том числе кластерной архитектуры.

Была модифицирована стадия построения промежуточных выравниваний по кластерному дереву. Параллелизм получен за счёт параллельного обхода дерева от листьев к корню. Для этого используется инструментальная система PARUS [8], которая позволяет записывать параллельную программу в нотации графа потока данных (граф-программа) (http://parus.sf.net). Здесь вершинам сопоставляются специальным образом оформленные функции языка C++, в которых происходят вычисления, а рёбрам сопоставляются операции передачи данных с помощью библиотеки MPI, или операции копирования памяти для случая, когда данные оказались на том же узле вычислительного кластера.

Предлагаемый алгоритм также, как и исходный, разделён на несколько стадий. На первой стадии используется оригинальный алгоритм MUSCLE для подсчёта матрицы схожести между последовательностями и построения кластерного дерева. На второй стадии

строится граф-программа, и производится её компиляция в C++ программу с вызовами функций MPI библиотеки. На третьей стадии производится исполнение полученной программы средствами PARUS. Для реализации 2-ой стадии написана программа, которой на вход подаётся кластерное дерево и файл с последовательностями в FASTA формате, а на выходе получается исходный код граф-программы в нотации PARUS.

Граф-программа строится по кластерному дереву следующим образом. Каждая вершина-исток в граф-программе соответствует одной или нескольким последовательностям. В программный код вершины вставляется вызов однопроцессорного построения множественного выравнивания для указанных последовательностей. В результате получается серия первых промежуточных выравниваний. Рёбра в граф-программе направлены от источников, которые соответствуют листьям кластерного дерева к внутренним вершинам, которые соответствуют внутренним вершинам кластерного дерева. Корень кластерного дерева является стоком в граф-программе. Внутренние вершины выравнивают пару промежуточных выравниваний, в результате получается новое промежуточное выравнивание. Для этого в код вершины вставляется вызов MUSCLE, который осуществляет выравнивание пары выравниваний. Это действие производится на одном из MPI-процессов, по возможности, параллельно с другими такими же построениями промежуточных выравниваний на других MPI-процессах. По достижении граф-программой корня кластерного дерева будет получено полное множественное выравнивание последовательностей.

Итак, граф-программа строится от листьев кластерного дерева к его корню. В случае, когда производится выравнивание множества коротких последовательностей, листьям кластерного дерева в граф-программе будет сопоставлен код, для которого накладные расходы на запуск и синхронизацию превысят положительный эффект от параллелизма. Для избежания такой ситуации в программу добавлен специальный параметр, который позволяет слить несколько уровней кластерного дерева в один уровень граф-программы. Именно по этой причине в вершине истоке граф-программы может оказаться несколько последовательностей, что исключено для листьев кластерного дерева.

4. Интернет-сервис

Интернет-сервис создан для повышения доступности использования многопроцессорной техники при решении своих практических задач молекулярным биологом. Сервис состоит из нескольких компонент: Веб-интерфейса, который создан на основе технологии PHP5+MySQL+Apache2; Python скрипта, который запускается через определённые промежутки времени и осуществляет передачу данных по SSH протоколу на многопроцессорную систему и обратно, а также осуществляет опосредованный запуск и остановку программных реализаций множественного выравнивания на многопроцессорной системе; bash скриптов, которые ставят задачу в очередь на многопроцессорной системе, просматривают её состояние и снимают со счёта в случае, когда это необходимо. По завершении задачи на многопроцессорной системе пользователю на его адрес электронной почты отправляется уведомление, в котором указан полученный статус задачи.

На начальном этапе пользователю интернет-сервиса предлагается зарегистрироваться на страничке регистрации пользователей, предоставляемой Веб-интерфейсом. (Регистрация осуществляется после указания пользователем адреса электронной почты и получения Веб-интерфейсом подтверждения). В дальнейшем пользователь может создать набор задач, где для каждой задачи выбирается: многопроцессорная система, число запрашиваемых процессоров, алгоритм множественного выравнивания, предполагаемая продолжительность решения задачи на многопроцессорной системе при указанных параметрах. После создания задачи пользователю предлагается загрузить через Веб-интерфейс данные в FASTA формате, либо напрямую через специальное поле ввода текста, либо, присоединив файл через соответствующую HTML форму.

Рис. 1. Граф переходов между значениями статуса задачи. Сплошной стрелкой обозначены переходы, которые пользователь может произвести самостоятельно. Разрывной стрелкой – переходы, которые делаются автоматически.

После загрузки последовательностей на сервер с Веб-интерфейсом пользователь может поменять статус своей задачи, тем самым создав заявку на исполнение. Для каждой задачи фиксируются дата её создания и дата завершения её исполнения на многопроцессорной технике. После завершения задачи пользователь может скачать результаты со специальной странички Веб-интерфейса и удалить более не нужные ему данные.

В текущий момент, через Интернет-сервис можно воспользоваться алгоритмами: ClustalW-MPI [4] и модифицированным MUSCLE. Параллельно выравнивать последовательности можно на машинах: mvs100к кластер на 982 узла с 2-мя четырёх-ядерными процессорами Intel Xeon 5160, кластере Чебышёв МГУ на 633 узла с 2-мя четырёх-ядерными процессорами Intel Xeon E5472, машине regatta 16-ти процессорной системе IBM pSeries 690 на общей памяти с процессорами Power4.

Остановимся более подробно на возможных значениях статуса задачи (см. рис. 1):

1. new – задача только что создана, в этом статусе позволено менять произвольные параметры задачи, в том числе загружать необходимые входные данные.

2. ready – задача находится в состоянии перемещения данных на многопроцессорную систему. Пользователь не может менять параметры задачи в этом статусе.

3. submited – задача поставлена в очередь на многопроцессорной системе. Пользователь не может менять параметры задачи.

4. finished – задача успешно завершена. Пользователь может менять параметры задачи, в том числе загружать новые данные.

5. refused – означает, что в процессе передачи данных на удалённую машину, или в процессе постановки задачи в очередь, или в процессе счёта произошла некоторая ошибка. Все промежуточные данные, в том числе журнал процесса счёта задачи, сохраняются в каталог задачи пользователя на веб-сервере. Пользователь может посмотреть нужные файлы и затем удалить их. В данном статусе разрешается менять параметры задачи.

6. stop – означает, что пользователь желает завершить данную задачу. Если задача переведена в статус stop, то в дальнейшем она может только перейти в статус refused.

Таблица 1. Время тестирования для LTR5

многопроцессорная система	последовательная версия (мин)	12 пр.	16 пр.
IBM pSeries 690	>420	21	24
PrimePOWER 850	68	28	-

Таблица 2. Время тестирования для аминокислотных последоаптельностей на машине mvs100k

последовательная версия (сек)	16 ядер	100 ядер	500 ядер
420	28	21	245

Переходы между различными значениями статуса разделены на 2-группы: ручные переходы и автоматические переходы. Ручной переход производится пользователем в момент, когда он хочет отправить задачу на счёт или остановить задачу. Автоматические переходы производятся с помощью Python скрипта в определённые промежутки времени (в текущей реализации один раз в час). О завершении процесса счёта с тем или иным статусом пользователь может узнать двумя способами: через Веб-интерфейс, а также прочитав свою электронную почту. На адрес электронной почты отправляется уведомление о завершении задачи.

Веб-интерфейс для Интернет сервиса доступен по адресу: http://angel.cs.msu.su/aligner.

5. Результаты тестирования data-flow алгоритма MUSCLE

Алгоритм был протестирован на последовательностях LTR (Long Terminal Repeat) класса 5 в геноме человека. LTR — семейство повторов (последовательность нуклеотидов, многократно повторяющаяся в геноме практически в неизменном виде); существует несколько классов таких повторов, классы отличаются своим происхождением с точки зрения биологической эволюции. Класс 5 (LTR5) содержит приблизительно 1500 последовательностей по 1200 нуклеотидов (коллекция всех LTR5 человеческого генома [7]). Было построено множественное выравнивание 1088 аминокислотных последовательностей приблизительно по 300 аминокислотных остатков каждая. В таблицах 1 и 2 приведены результаты тестирования для машин PrimePOWER 850 с 12 процессорами, IBM pSeries 690 и mvs100k. В таблице 3 приведены усреднённые значения ускорения по всем тестам и всем машинам.

Время исполнения параллельной программы существенно зависит от нескольких параметров. В первую очередь оно сильно зависит от сбалансированности кластерного дерева, что влияет на число вершин, которое приходится на один уровень граф-программы (уровень в граф-программе — уровень в ориентированном лесе); зависит от числа, объёма и структуры промежуточных выравниваний, приходящихся на вершину граф-программы. Время зависит от числа и производительности процессоров, на которых запущена полученная MPI-программа, а также довольно сильно от объёма памяти, приходящейся на узел

Таблица 3. Среднее ускорение по всем тестам на всех машинах

число проц (ядер)	12	16	100	500
ускорение (разы)	2,4	17,5	20	1,7

вычислительного кластера.

Как видно из результатов тестирования, для эффективного постороения выравнивания на многопроцессорной системе по предложенному алгоритму, оказывается важно подобрать правильное число процессоров для счёта. К сожалению алгоритм обладает не очень хорошей масштабируемостью. Дальнейшая работа будет предполагать улучшение масштабируемости за счёт более глубокого пересмотра реализации стадий алгоритма MUSCLE.

Литература

1. Saul B. Needleman and Christian D. Wunsch A general method applicable to the search for similarities in the amino acid sequence of two proteins //Journal of Molecular Biology Volume 48, Issue 3, 1970, pp. 443-453, ISSN: 0022-2836.

2. T.F. Smith, M.S. Waterman and W.M. Fitch Comparative biosequence metrics //Journal of Molecular Evolution Volume 18, Number 1, 1982, pp. 38-46, ISSN: 0022-2844 (Print), ISSN: 1432-1432 (Electronic).

3. Julie D. Thompson, Desmond G. Higgins, Toby J. Gibson CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice //Nucleic Acids Research, 1994, vol. 22 No. 22 , pp. 4673-4680. ISSN: 0305-1048 (Print), ISSN: 1362-4962 (Electronic).

4. Kuo-Bin Li ClustalW-MPI: ClustalW analysis using distributed and parallel computing //Bioinformatics Vol. 19, No. 12, 2003, pp. 1585-1586. ISSN: 1460-2059 (Electronic),ISSN: 1367-4803 (Print)

5. Robert C. Edgar MUSCLE: a multiple sequence alignment method with reduced time and space complexity //BMC Bioinformatics 2004, 5:113, ISSN: 1471-2105 (Electronic).

6. P.H.A. Sneath, Robert R. Sokal Numerical Taxonomy //Nature 193, pp. 855-860 (03 March 1962), ISSN: 0028-0836, EISSN: 1476-4687.

7. Alexeevski A.V., Lukina E.N., Salnikov A.N., Spirin S.A. Database of long terminal repeats in human genome: structure and synchronization with main genome archives //Proceedings of the fours international conference on bioinformatics of genome regulation and structure, Volume 1. BGRS 2004, pp. 28-29 Novosibirsk.

8. Alexey N. Salnikov PARUS: A Parallel Programming Framework for Heterogeneous Multiprocessor Systems //Lecture Notes in Computer Science (LNCS 4192) Recent Advantages in Parallel Virtual Machine and Message Passing Interface, Volume 4192, pp. 408-409, 2006, ISBN-10: 3-540-39110-X ISBN-13: 978-3-540-39110-4.

9. Amarendran R Subramanian, Michael Kaufmann and Burkhard Morgenstern DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment // Algorithms for Molecular Biology, 2008, 3:6, ISSN: 1748-7188 (Электронный).

10. Martin Schmollinger, Kay Nieselt, Michael Kaufmann and Burkhard Morgenstern DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors // BMC Bioinformatics, 2004, 5:128, ISSN: 1471-2105 (Электронный).