Математическая биология и биоинформатика
2017. Т. 12. № 1. С. 137–150. doi: 10.17537/2017.12.137

========== **БИОИНФОРМАТИКА** ==========

УДК: 575.112:004

Параллельный алгоритм глобального выравнивания протяжённых аминокислотных и нуклеотидных последовательностей

©2017 Тетуев Р.К., Пятков М.И., Панкратов А.Н.*

Институт математических проблем биологии РАН – филиал ИПМ им. М.В.Келдыша РАН, Пущино, Московская область, Россия

Аннотация. Разработан параллельный алгоритм для глобального выравнивания протяженных последовательностей. Алгоритм использует произвольную матрицу замен. Аффинная система штрафов за внутренние и концевые разрывы в выравнивании может быть задана раздельно для каждой последовательности. Реализована возможность управления выбором оптимального выравнивания из множества альтернативных. Параметрами параллельного алгоритма являются шаги сетки, которая разбивает матрицу глобального выравнивания на блоки. Проведены исследования и выработаны критерии выбора этих параметров как для оптимизации использования памяти, так и по сокращению времени работы алгоритма. Показано, что при выборе размеров блоков, обеспечивающих оптимизацию сложности по памяти, алгоритм позволяет выравнивать протяженные последовательности длины L, используя объем памяти $O(L^{4/3})$. Дополнительно показано, что алгоритм идеально масштабируется на многоядерных системах, демонстрируя суперлинейное ускорение. Алгоритм реализован в виде высокопроизводительного параллельного веб-приложения на языке JavaScript, доступного по адресу http://sbars.impb.ru/aligner.html.

Ключевые слова: *глобальное выравнивание, аффинная система штрафов, концевые вставки, параллельные вычисления, суперлинейное ускорение.*

ВВЕДЕНИЕ

В настоящее время методы биоинформатики активно развиваются в связи с совершенствованием технологий секвенирования последовательностей и возникновением больших объёмов данных. Новые возможности открылись с появлением облачных вычислений, суть которых заключается в предоставлении пользователю удалённого доступа к вычислительным мощностям или специализированному программному обеспечению [1, 2]. Примером является сервис BLAST [3], который предоставляет различные инструменты сравнения и поиска нуклеотидных и аминокислотных последовательностей.

В данной работе рассматривается классический алгоритм биоинформатики – алгоритм Нидлмана-Вунша [4] для глобального выравнивания нуклеотидных и аминокислотных последовательностей. Целью данной работы является разработка инструмента для оценки близости протяжённых повторов в геномах, обнаруживаемых с помощью спектрально-аналитического метода [5, 6].

*pan@impb.ru

Как правило, современные реализации алгоритма глобального выравнивания NCBI BLAST [7] и EMBOSS [8] учитывают аффинную систему штрафов за вставки, предложенную в работе [9], а также систему штрафов за вставки на концах последовательностей [10]. В то же время параметры метода в этих реализациях не всегда можно поменять в желаемых пределах, по-разному решаются вопросы выбора уникального выравнивания из множества альтернативных. Таким образом, каждая реализация имеет свои особенности, которые не позволяют сравнивать ее в точности с другими. Однако основное ограничение этих реализаций – на длину исследуемых последовательностей, как правило, не более нескольких десятков тысяч нуклеотидов. Это связано с тем, что алгоритм глобального выравнивания является квадратичным как по памяти, так и по быстродействию в зависимости от длины обрабатываемых последовательностей.

Преодолению сложности алгоритма по памяти посвящена работа [11], в которой предложен алгоритм, который понижает потребление памяти до линейного за счёт рекурсивного подхода, предложенного ранее в работе [12] для задачи поиска общей подстроки максимальной длины. Суть этого подхода заключается в том, что одна из последовательностей делится пополам и находится соответствующая точка во второй последовательности, через которую проходит оптимальное выравнивание. После этого задача рекурсивно сводится к выравниванию двух пар фрагментов и т.д. При этом теоретическая вычислительная сложность удваивается по сравнению с классическим алгоритмом, в котором матрица динамического программирования сохраняется в памяти.

Несмотря на появление и развитие более быстрых эвристических алгоритмов для исследования генетических последовательностей [13], тема глобального выравнивания продолжает развиваться, имеются современные подходы его оптимизации [14, 15, 16, 17, 18], а также реализации для новых компьютерных архитектур [19, 20]. Среди этих работ особое место занимает работа [14], в которой предложена сеточная схема алгоритма глобального выравнивания. В сеточной схеме матрица динамического программирования разбивается на блоки, память выделяется под значения элементов матрицы на границах блоков, а значения элементов матрицы внутри блоков могут быть вычислены, исходя из граничных. Вычисления экономятся за счёт того, что при обратном проходе матрицы динамического программирования восстановлению подлежит небольшое число блоков. Показано, что в случае рекурсивного применения этот подход также понижает использование памяти до линейного.

В данной работе предлагается наиболее общая параллельная реализация алгоритма Нидлмана-Вунша. За основу взята сеточная схема алгоритма, которая позволяет как экономить память, так и распараллеливать вычисления. Однако в отличии от работ [11, 14], предложивших алгоритмы с линейным ростом памяти, в данной работе определяется минимум требуемой памяти как баланс между её использованием при прямом и обратном проходе метода. Показано, что, хотя рост памяти в этом случае больше, чем линейный, это позволяет выравнивать последовательности до миллиона нуклеотидов, т.е. на два порядка длиннее, чем стандартными методами. При этом разработанный алгоритм характеризуется идеальной масштабируемостью вычислений на многоядерных вычислительных системах.

ПОСТАНОВКА ЗАДАЧИ

Пусть $Q = (Q_1, \ldots, Q_{L_Q})$ и $S = (S_1, \ldots, S_{L_S})$ некоторая пара строк, составленных из символов алфавита $Q_i, S_j \in A \equiv \{A_1, \ldots, A_N\}$ при $1 \leqslant i \leqslant L_Q$ и $1 \leqslant j \leqslant L_S$. Пусть далее

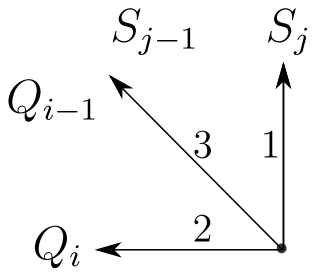

Рис. 1. Графическое представление элементарных редакций.

определены элементарные редакции $t(Q_i, S_j)$ трёх видов [21] со следующими весами:

$$\|t(Q_i, S_j)\| = \begin{cases} G^S & \text{– вставка разрыва перед } S_{j+1} \\ G^Q & \text{– вставка разрыва перед } Q_{i+1} \\ W_{Q_i,S_j} & \text{– соответствие символов: } Q_i \leftrightarrow S_j, \end{cases} \tag{1}$$

где веса замен заданы матрицей [22, 23]:

$$W = \begin{pmatrix} W_{A_1,A_1} & W_{A_1,A_2} & \cdots & W_{A_1,A_N} \\ W_{A_2,A_1} & W_{A_2,A_2} & \cdots & W_{A_2,A_N} \\ \vdots & \vdots & \ddots & \vdots \\ W_{A_N,A_1} & W_{A_N,A_2} & \cdots & W_{A_N,A_N} \end{pmatrix}, \tag{2}$$

а веса G^Q, G^S выбираются из набора

$$G = \{p_Q, P_Q, p_Q^L, P_Q^L, p_Q^R, P_Q^R, p_S, P_S, p_S^L, P_S^L, p_S^R, P_S^R\} \tag{3}$$

в зависимости от контекста возникновения вставки: заглавные буквы P соответствуют весам открывающих вставок, строчные буквы p – весам протяжённых вставок, верхние индексы L и R – весам за концевые вставки слева или справа, а нижний индекс указывает на последовательность. На рисунке 1 схематично изображены элементарные редакции на плоскости $i \times j$ под номерами, соответствующими положению в списке (1).

Используя некоторый ряд элементарных правок $T(Q, S)$, можно сравнить строку Q со строкой S с суммарным весом выравнивания:

$$\|T(Q, S)\| = \sum_{k=1}^{K} \|t(Q_{i_k}, S_{j_k})\|, \tag{4}$$

где $K \leqslant (L_Q + L_S)$ – количество элементарных правок, а для всех индексов i_k, j_k соблюдается условие упорядоченности правок: $i_k \leqslant i_{k+1}$ и $j_k \leqslant j_{k+1}$. Пусть $\tau = \{T(Q, S)\}$ – множество всех оптимальных редакций, для которых (4) принимает максимальное значение, а F – некоторое правило выбора уникальной редакции:

$$T^* = F(\tau(Q, S; W, G)). \tag{5}$$

Задачей данной работы является построение параллельного алгоритма решения для нахождения уникального оптимального выравнивания T^* двух строк Q, S при произвольной матрице замен W, наборе параметров G и правиле отбора F.

АЛГОРИТМИЧЕСКОЕ РЕШЕНИЕ

Выделив последнюю элементарную редакцию в (4)

$$\|T(Q,S)\| = \sum_{k=1}^{K} \|t(Q_{i_k}, S_{j_k})\| = \sum_{k=1}^{K-1} \|t(Q_{i_k}, S_{j_k})\| + \|t(Q_{i_K}, S_{j_K})\|, \tag{6}$$

можно составить рекуррентное соотношение

$$\|T(Q,S)\| = \max \begin{pmatrix} \|T(Q^{\,i-1}, S^{\,j}\ )\| + G^S, \\ \|T(Q^{\,i}\ , S^{\,j-1})\| + G^Q, \\ \|T(Q^{\,i-1}, S^{\,j-1})\| + W_{Q_i,S_j} \end{pmatrix}, \tag{7}$$

где $Q^{\,i-1} = (Q_1, \ldots, Q_{i-1})$ и $S^{\,j-1} = (S_1, \ldots, S_{j-1})$ – подстроки Q и S при $i = L_Q$ и $j = L_S$. Из трёх величин, из которых находится максимум, можно составить вектор выбора вариантов:

$$V_{i,j} = (V_{i,j,1}, V_{i,j,2}, V_{i,j,3}).$$

Причём хотя бы один из этих трёх вариантов из последнего шага редакционного ряда совпадает с наилучшим значением веса выравнивания:

$$\|T(Q,S)\| = V_{i,j}^{max} = \max(V_{i,j,1}, V_{i,j,2}, V_{i,j,3}).$$

Теперь можно переписать рекуррентное соотношение (7) в виде:

$$V_{i,j}^{max} = \max \begin{pmatrix} V_{i-1,j}^{max} & + G^S, \\ V_{i,j-1}^{max} & + G^Q, \\ V_{i-1,j-1}^{max} & + W_{Q_i,S_j} \end{pmatrix}. \tag{8}$$

Формула (8) задаёт рекуррентное соотношение для $V_{i,j}^{max}$ при $0 \leqslant i \leqslant L_Q$ и $0 \leqslant j \leqslant L_S$, начальным состоянием которого является случай двух пустых строк:

$$V_{0,0}^{max} \equiv \|T(Q^0, S^0)\| = 0. \tag{9}$$

В рекуррентной формуле (8) используются штрафы за вставки G^S и G^Q, которые можно определить для случая линейной модели штрафов с концевыми штрафами:

$$G_i^Q = \begin{cases} P_Q^L & i = 0 \\ P_Q & 0 < i < L_Q \\ P_Q^R & i = L_Q, \end{cases} \qquad G_j^S = \begin{cases} P_S^L & j = 0 \\ P_S & 0 < j < L_S \\ P_S^R & j = L_S. \end{cases} \tag{10}$$

В случай аффинной модели дополнительно вводятся штрафы за протяжённые вставки: $p^Q \neq P^Q$ и $p^S \neq P^S$, для которых также имеет место разграничение между концевыми и внутренними вставками с определением, аналогичным (10). Тогда первый элемент рекуррентного соотношения (8) имеет вид:

$$V_{i,j,1} = V_{i-1,j}^{max} + G_j^{\,S} = \max \begin{pmatrix} V_{i-1,j,1} + G_j^{\,S}, \\ V_{i-1,j,2} + G_j^{\,S}, \\ V_{i-1,j,3} + G_j^{\,S} \end{pmatrix} = \max \begin{pmatrix} V_{i-2,j}^{max} & + G_j^{\,S} & + G_j^{\,S}, \\ V_{i-1,j-1}^{max} & + G_{i-1}^{\,Q} & + G_j^{\,S}, \\ V_{i-2,j-1}^{max} & + W_{Q_{i-1},S_j} & + G_j^{\,S} \end{pmatrix}.$$

Очевидно, что вставка G_j^S является дублирующей после вставки того же типа варианта $V_{i-2,j,1}$, а значит эта вставка является протяжённой и здесь наказывается меньшим

Математическая биология и биоинформатика. 2017. Т. 12. № 1. doi: 10.17537/2017.12.137

штрафом: $G_j^S = p_S^R$. Для второго элемента ситуация обратная – вставке G_j^S предшествует вставка в другую строку G_i^Q, но в этом случае вставка должна быть наказана бóльшим штрафом, как первая, открывающая вставка: $G_j^S = P_S^R$. Ясно, что для последнего варианта любая последующая вставка является открывающей. После проведения подобных построений для всех трёх вариантов формулы (8) получается соотношение:

$$V_{i,j} = (V_{i,j,1}, V_{i,j,2}, V_{i,j,3}) = \left(\max \begin{pmatrix} V_{i-1,j,1} + p_j^S, \\ V_{i-1,j,2} + P_j^S, \\ V_{i-1,j,3} + P_j^S \end{pmatrix}, \max \begin{pmatrix} V_{i,j-1,1} + P_i^Q, \\ V_{i,j-1,2} + p_i^Q, \\ V_{i,j-1,3} + P_i^Q \end{pmatrix}, \max \begin{pmatrix} V_{i-1,j-1,1} + W_{Q_i,S_j}, \\ V_{i-1,j-1,2} + W_{Q_i,S_j}, \\ V_{i-1,j-1,3} + W_{Q_i,S_j} \end{pmatrix} \right). \tag{11}$$

Для линейной модели достаточно реализовать формулу (8), возвращающую единственное, максимально оптимальное значение $V_{i,j}^{max}$, полученное на предыдущем шаге рекурсии. Аффинная модель (11) требует рассмотрения целого вектора $V_{i,j}$, составленного из трёх возможных шагов редакции. За счёт этого удаётся ослабить штрафы за текущие вставки, если их тип совпадает с типом предыдущей операции.

Значения вектора $V_{i,j}$, вычисленные по рекуррентной формуле (11), заполняют таблицу размера $(L_Q + 1) \times (L_S + 1)$:

$$V = \begin{pmatrix} V_{0,0} & V_{0,1} & \cdots & V_{0,L_S} \\ V_{1,0} & V_{1,1} & \cdots & V_{0,L_S} \\ \vdots & \vdots & \ddots & \vdots \\ V_{L_Q,0} & V_{L_Q,1} & \cdots & V_{L_Q,L_S} \end{pmatrix}. \tag{12}$$

Согласно (9) в таблице V верхнее угловое значение $V_{0,0} = (0, 0, 0)$ вне зависимости от содержания и длины рассматриваемых строк Q и S. Значение в соседней правой ячейке также не зависит от строк и всегда равно $V_{0,1}^q = P_Q^L$, т.к. оно соответствует первой вставке в строку Q на крайне левом её конце. В следующей ячейке должно находиться значение, соответствующее штрафу не за одну, а за две такие последовательные вставки, что в общем виде равно сумме открывающего и протяжённого штрафов: $V_{0,1}^q = P_Q^L + p_Q^L$. Продолжая этот ряд, получается общее выражение: $V_{0,j}^q = P_Q^L + (j-1)\, p_Q^L$ при $1 \leqslant j \leqslant L_S$. Аналогично, для первого столбца таблицы V получается выражение: $V_{i,0}^s = P_S^L + (i-1)\, p_S^L$ при $1 \leqslant i \leqslant L_Q$. Таким образом, ячейки таблицы вычисляются по рекуррентной формуле (11) при наличии следующих начальных значений:

$$V_{i,j} = (V_{i,j,1}, V_{i,j,2}, V_{i,j,3}) = \begin{cases} (0, 0, 0) & i = 0, j = 0 \\ (-\infty,\ P_Q^L + (j-1)\, p_Q^L,\ -\infty) & i = 0, 1 \leqslant j \leqslant L_S \\ (P_S^L + (i-1)\, p_S^L,\ -\infty,\ -\infty) & 1 \leqslant i \leqslant L_Q, j = 0, \end{cases} \tag{13}$$

где $-\infty$ соответствует отсутствующим вариантам рекурсии в данной позиции [10]. Бесконечный штраф вводит запрет на вставку в последовательность, что может быть использовано для оценивания расстояния Хэмминга [24].

Суммарная оценка (4) для оптимальной редакции (5) рассматриваемых строк Q и S

находится в последнем векторе V_{L_Q,L_S} и равна максимальному из трёх компонент:

$$\|T^*(Q,S;W,G)\| = V^{max}_{L_Q,L_S}.$$

Следовательно, если от алгоритма требуется только соответствующая оценка сходства для данной пары строк, то конечный результат получен в последней ячейке таблицы векторов. Однако в общей постановке задачи требуется найти некоторый оптимальный редакционный ряд, притом единственный, строго определённый из множества всех альтернативных оптимальных редакций.

Максимальному весу в каждой компоненте вектора (11) соответствует элементарная редакция, которая приводит к этому весу и которую следует запомнить для построения выравнивания. Таким образом, наряду с вектором весов следует ввести вектор элементарных редакций, при которых достигаются максимальные значения весов

$$Z_{i,j} = (Z_{i,j,1}, Z_{i,j,2}, Z_{i,j,3}), \tag{14}$$

каждая из компонент которого принимает значения 1, 2 или 3, в зависимости от того, какая из компонент векторов в формуле (11) принимает максимальное значение.

Если на некотором шаге рекурсии возникает ситуация, когда два или три варианта имеют равные максимальному значения, то это приводит к неоднозначности выбора. С точки зрения веса выравнивания произвольный выбор любого из равных вариантов никак не влияет на конечный результат, но с точки зрения хода выравнивания выбор одного из альтернативных оптимальных вариантов на каждом шаге задаёт самостоятельную уникальную редакцию. При этом множество альтернативных редакций $\tau(Q,S;W,G)$ многократно растёт с каждым шагом, на котором возникает неоднозначность выбора варианта. Разрешение неоднозначности на каждом шаге может быть осуществлено с помощью правила приоритета.

Пусть, к примеру, некоторое правило F_{123} задаёт условие приоритета такое, что альтернативный выбор между операцией вставки в строку S (операция 1) и другими оптимальными операциями всегда разрешается в пользу первой, а операция вставки в строку Q (операция 2) имеет больший приоритет над равным альтернативным значением для операции установления соответствия (операция 3). В отличие от произвольных правил, предлагаемый критерий является *контекстным*, т.к. выбор единственного, уникального редакционного ряда производится на уровне отдельного шага рекурсии каждый раз при возникновении неоднозначности. Всего контекстных правил с заранее определённым приоритетом шесть:

$$F \in \{F_{123}, F_{213}, F_{132}, F_{231}, F_{312}, F_{321}\}.$$

Вообще, помимо правила приоритетов, на практике можно предложить и другие критерии выбора уникального оптимального выравнивания из множества всех альтернативных, к примеру, случайный выбор или вывод всех альтернативных выравниваний [25, 26]. Правило F_{123} применяется в сервисе глобального выравнивания NCBI BLAST [7], а более сложное правило, не из списка элементарных контекстных правил, применяется в сервисе глобального выравнивания EMBOSS [8].

При условии определения матрицы (14) редакционный ряд, соответствующий

максимальному компоненту веса $V^{max}_{L_Q,L_S}$, восстанавливается следующим образом:

$$\begin{aligned} z_{k-1} &= Z_{i_k,j_k,z_k}, \quad k = K, ..., 2 \\ i_{k-1} &= i_k - 1, \quad \text{если} \quad z_k \in \{1, 3\} \\ j_{k-1} &= j_k - 1, \quad \text{если} \quad z_k \in \{2, 3\}, \end{aligned} \tag{15}$$

где K – длина искомого редакционного ряда (5), а сам ряд – перевёрнутая строка индексов

$$(z_1, z_2, \ldots, z_{K-1}, z_K), \tag{16}$$

задающих тип операций, и соответствующая искомой уникальной редакции.

Таким образом, алгоритм состоит из двух основных этапов:

1) *прямой проход*: вычисление весов (12) и соответствующих им элементарных редакций (14) по рекуррентной формуле (11);

2) *обратный проход*: восстановление лучшей редакции по рекуррентному соотношению (15) с использованием таблицы элементарных редакций (14).

РАСПАРАЛЛЕЛИВАНИЕ АЛГОРИТМА

Для всех случаев, когда специфика практической задачи требует сравнения протяжённых строк, порядка нескольких тысяч символов, требования к оперативной памяти становятся слишком высокими. Задачу тем не менее можно решить, если разбить таблицу на небольшие прямоугольные блоки, каждый из которых свободно умещается в оперативной памяти. При этом нет необходимости держать в памяти все блоки одновременно, а достаточно одного блока таблицы, по ячейкам которого строится текущий участок обратного прохода. Когда во время итерации обратный проход выходит за рамки загруженного блока таблицы, в оперативной памяти следует восстановить следующий блок, в котором производятся текущие вычисления.

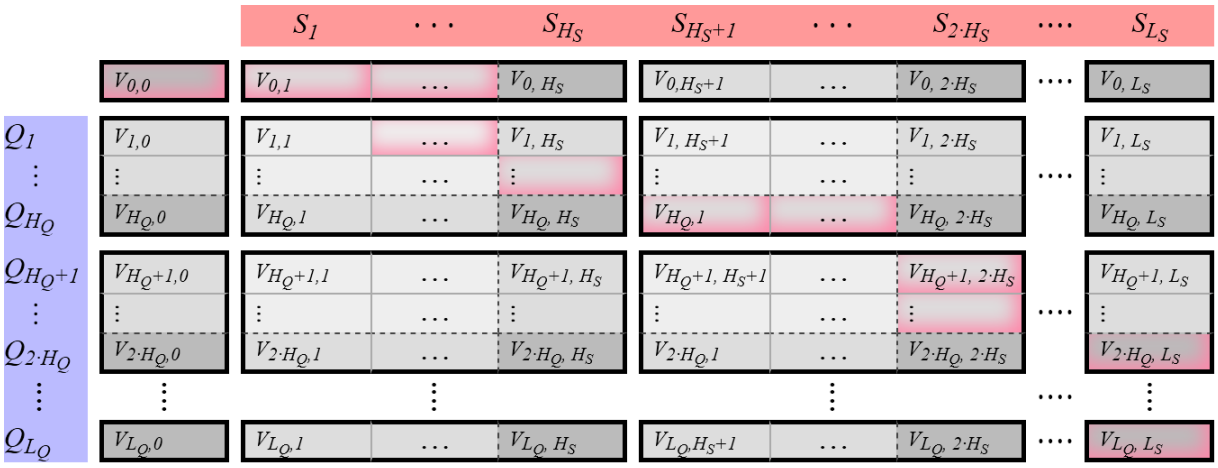

Рис. 2. Таблица векторов выбора с разбиением на блоки. Розовым цветом обозначены ячейки предполагаемого обратного прохода.

Пусть таблица разделена на блоки посредством разбиения строки Q на фрагменты вида:

$$(Q_1, Q_2, \ldots), (Q_{H_Q+1}, Q_{H_Q+2}, \ldots), (Q_{2H_Q+1}, Q_{2H_Q+2}, \ldots), \ldots, (Q_{L_Q}), \tag{17}$$

а строки S на фрагменты

$$(S_1, S_2, \ldots), (S_{H_S+1}, S_{H_S+2}, \ldots), (S_{2H_S+1}, S_{2H_S+2}, \ldots), \ldots, (S_{L_S}), \tag{18}$$

где H_Q – высота внутренних блоков таблицы V, H_S – ширина внутреннего блока. Тогда каждой паре фрагментов строк Q, S сопоставлен некоторый блок $U_{I,J}$ таблицы для пары неотрицательных индексов $I, J \geqslant 0$:

$$U_{I,J} = \begin{pmatrix} V_{IH_Q+1,JH_S+1} & V_{IH_Q+1,JH_S+2} & \cdots & V_{IH_Q+1,JH_S+H_S} \\ V_{IH_Q+2,JH_S+1} & V_{IH_Q+2,JH_S+2} & \cdots & V_{IH_Q+2,JH_S+H_S} \\ \vdots & \vdots & \ddots & \vdots \\ V_{IH_Q+H_Q,JH_S+1} & V_{IH_Q+H_Q,JH_S+2} & \cdots & V_{IH_Q+H_Q,JH_S+H_S} \end{pmatrix}. \tag{19}$$

Начальные ячейки таблицы (12) не войдут ни в один из блоков (19), а составят блоки с отрицательными индексами $I = -1$, $J = -1$ со значениями согласно формуле (13):

$$\begin{cases} U_{-1,-1} = \big((0,0,0)\big) \\ U_{-1,J} = \big((-\infty, P_Q^L{+}JH_S\, p_Q^L, -\infty), (-\infty, P_Q^L{+}(JH_S{+}1)\, p_Q^L, -\infty), \ \ldots\big), & 0 \leqslant J \\ U_{I,-1} = \big((P_S^L{+}IH_Q p_S^L, -\infty, -\infty), (P_S^L{+}(IH_Q{+}1)\, p_S^L, -\infty, -\infty), \ \ldots\big)^\top, & 0 \leqslant I. \end{cases} \tag{20}$$

Аналогично начальным блокам (20) рационально выделить конечные блоки, соответствующие последнему символу каждой строки (17, 18). Благодаря этому техническому приёму, для каждого блока (12) задан строго определённый тип штрафов: внутренние $\{p_Q, P_Q\}, \{p_S, P_S\}$ или концевой правый $\{p_Q^R, P_Q^R\}, \{p_S^R, P_S^R\}$, и эти штрафы не меняются, постоянны на всём протяжении расчётов для этого блока также, как для блоков начальных значений (20) используются только концевые левые штрафы. При этом обязательная проверка достижения конца строки, требуемая для выбора определённого типа штрафа (10), проводится однократно, ещё до запуска расчёта конкретного блока, а не многократно на каждом шаге рекуррентной формулы (11). Это сказывается на общей скорости алгоритма за счёт исключения условных операций (10) из внутреннего цикла рекуррентных вычислений (11).

Значения любого блока $U_{I,J}$ вида (19) можно заполнить по рекуррентной формуле (11) тогда и только тогда, когда известны значения в трёх соседних от него блоках: $U_{I-1,J-1}$, $U_{I,J-1}$ и $U_{I-1,J}$, причём необходимы не все значения в этих блоках, а лишь в тех ячейках, которые непосредственно прилегают к блоку $U_{I,J}$. Таким образом, для заполнения блока вида (19) требуется задание двух векторов и одного скалярного значения из ячеек таблицы (12), которые можно объединить в виде следующих значений:

$$u_{I,J} = \begin{cases} u_{I-1,J-1}^{\text{axe}} = V_{IH_Q,JH_S} \\ u_{I-1,J}^{\text{row}} = (V_{IH_Q,JH_S+1}, V_{IH_Q,JH_S+2}, \ldots, V_{IH_Q,(J+1)H_S}) \\ u_{I,J-1}^{\text{col}} = (V_{IH_Q+1,JH_S}, V_{IH_Q+2,JH_S}, \ldots, V_{(I+1)H_Q,JH_S})^\top. \end{cases} \tag{21}$$

Пусть множество $u = \{u_{I,J}\}$ состоит из всех комплексов (21), вычисленных для каждого блока вида (19), тогда оно определяет все внутренние значения (19) согласно рекуррентной формуле (11). Можно сделать важный вывод, что множество u является сеточным представлением таблицы V, где скаляры $u_{I,J}^{\text{axe}}$ и векторы $u_{I,J}^{\text{row}}$, $u_{I,J}^{\text{col}}$ являются узлами и рёбрами этой сетки соответственно. Тогда задачу расчёта таблицы векторов выбора V можно заменить эквивалентной задачей вычисления сетки u, т.к. любой фрагмент таблицы при необходимости можно восстановить за сравнительно небольшое количество вычислений. Сама сетка при этом занимает значительно меньший объём памяти и может быть размещена в оперативной памяти целиком.

Сеточное представление позволяет не только поместить всю таблицу векторов выбора в оперативную память, но и многократно ускорить процесс на многопроцессорных

системах вычислений. Действительно, каждый блок таблицы можно вычислить тогда, когда для него заданы все собственные начальные значения. Отсюда следует очевидный вывод: блоки таблицы, для которых заданы соответствующие соседние рёбра и узлы сетки u, могут быть рассчитаны независимо и одновременно, т.е. параллельно.

РЕЗУЛЬТАТЫ

Программная реализация

Для достижения максимальной переносимости и масштабируемости вычислений к разрабатываемой программе предъявлены требования по части времени её исполнения, объёма занятой памяти, а также кросс-платформенной совместимости, т.е. возможности использования данной программы на всех платформах без необходимости переписывать и адаптировать исполняемый код. Благодаря многоступенчатой оптимизации алгоритма оказалось возможным разработать высокопроизводительное приложение на языке программирования JavaScript с необходимым интерфейсом пользователя на языке разметки HTML.

Для многократного ускорения кода и снижения системных требований были применены следующие подходы структурной оптимизации:

1) Замена векторов и матриц на типизированные массивы.

2) Разбиение матрицы динамического программирования на блоки, что позволяет рассчитывать блоки параллельно.

3) Применение таблиц поиска и битовых операций вместо операторов ветвления и вызовов функций, что позволяет сокращать время вычислений за счёт конвейеризации вычислений на уровне процессора.

Дальнейшее исследование касается параметрической оптимизации. Эффективность алгоритма по времени выполнения или по занимаемой памяти зависит от параметров алгоритма – шагов сетки H_Q и H_S. При этом существенный вклад в эффективность алгоритма по времени вносит масштабируемость параллельных вычислений, т.е. ускорение, достигаемое за счет распараллеливания.

Масштабируемость алгоритма на многоядерных системах

Объем необходимых вычислений разработанного алгоритма пропорционален количеству элементов матрицы динамического программирования с учетом повторяющихся вычислений на обратном проходе

$$O(L^2 + \frac{L}{H}H^2) = O(L^2 + LH), \tag{22}$$

где $L = L_Q = L_S$ – длина последовательности, причём обе последовательности имеют одинаковую длину, $H = H_Q = H_S$ – размер одного блока, который также для простоты считается квадратным. Эта формула состоит из двух слагаемых: первое отвечает сложности прямого прохода, а второе – обратного прохода. Формула отражает тот факт, что на обратном проходе алгоритма требуется восстанавливать значения матрицы не во всех блоках, а только в L/H блоках размера $H \times H$.

Исследование эффективности распараллеливания алгоритма проводится эмпирически с помощью замеров производительности некоторой задачи на разном количестве процессоров. В таблице 1 приведены времена расчетов на персональном компьютере с четырёхядерным процессором AMD Phenom в интернет-браузере FireFox под

управлением операционной системы Windows для случайных последовательностей длины $L = 50000$.

Таблица 1. Время выполнения алгоритма, его параллельной (прямой проход) и непараллельной (обратный проход) частей при различных значениях размера блока H и числа процессоров M. Жирным шрифтом выделено лучшее время в каждой из колонок

H	M	Общее время, с	Прямой проход	Обратный проход
500	1	460.99	452.71	8.28
	2	241.92	233.85	8.07
	3	152.79	145.14	**7.65**
	4	88.76	80.55	8.21
1000	1	265.84	254.29	11.55
	2	145.84	135.2	10.64
	3	93.15	82.95	10.19
	4	**58.24**	**49**	9.24
5000	1	252.3	218.46	33.83
	2	159.23	125.31	33.92
	3	119.69	85.79	33.89
	4	90.88	56.7	34.17

Из таблицы видно, что сложность обратного прохода возрастает при увеличении размера сетки H, а сложность прямого прохода убывает с ростом H. В первом случае это находится в соответствии с формулой (22). Во втором случае это связано с накладными расходами при увеличении узлов сетки, которые не учитываются формулой. Поэтому для исследования ускорения в зависимости от H при увеличении числа процессоров M, необходимо рассматривать только время исполнения прямого прохода, который полностью распараллеливается.

На рисунке 3 показано ускорение A алгоритма для прямого прохода. Ускорение определяется как отношение времени выполнения алгоритма на одном процессоре ко времени выполнения на M процессорах. Из рисунка видно, что при значении $H = 1000$ получается лу́чшее ускорение алгоритма, чем при $H = 5000$. Причём это ускорение больше, чем теоретически максимальное линейное ускорение. Такое ускорение называется суперлинейным и объясняется использованием кэш-памяти процессоров. Эффект суперлинейного ускорения настолько выражен, что имеет место даже в случае анализа полного времени выполнения программы, указанного в таблице 1. При ме́ньшем размере блока H ускорение возрастает, но общая производительность при этом снижается вследствие накладных расходов при распараллеливании вычислений. Косвенным признаком этого служит то, что загрузка процессора не достигает максимальной несмотря на наличие достаточного количества параллельно работающих потоков. При бо́льшем размере блока эффективность программы снижается за счёт того, что кэш-память используется менее эффективно. Таким образом, оптимальный размер блока может быть найден эмпирически, исходя из лучшей производительности вычислений на многоядерных процессорах.

Минимизация использования памяти

При распараллеливании вычислений обычно сталкиваются с проблемой узкого горлышка, когда потеря эффективности происходит из-за возрастающего обмена процессоров с памятью. Тогда для повышения масштабируемости требуется менять

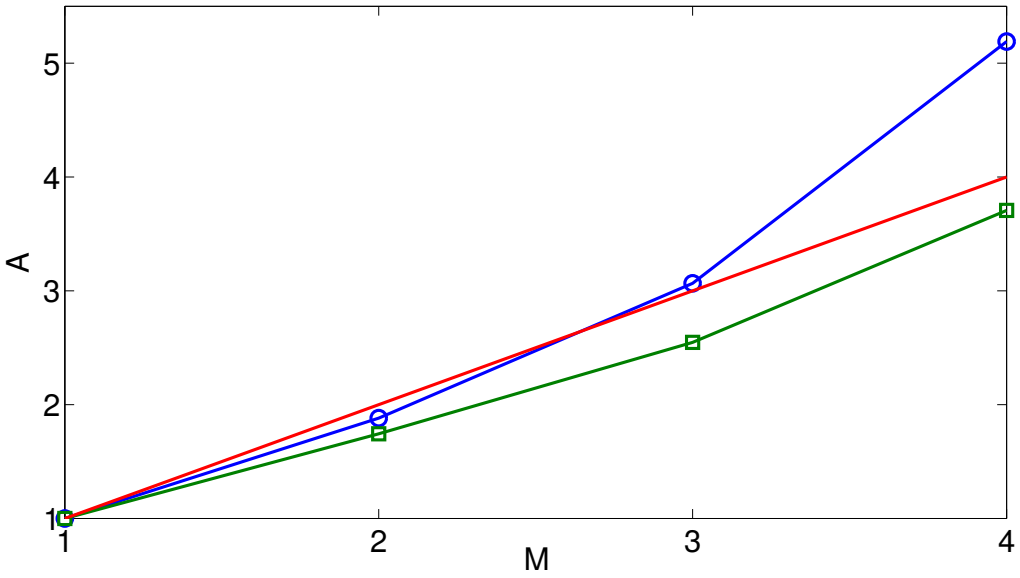

Рис. 3. Ускорение вычислений в зависимости от числа процессоров M при $H = 1000$ (круглые маркеры) и $H = 5000$ (квадратные маркеры) в сравнении с линейным ускорением (без маркеров).

алгоритм в сторону минимизации обращений к памяти. В ряде случаев можно добиться снижения обращений к памяти за счёт повторяющихся вычислений. При прямом проходе алгоритм оставляет значения вычисленной функции в памяти только в точках сетки для того, чтобы можно было восстановить значения для всех внутренних точек каждого блока матрицы на обратном проходе алгоритма. Поскольку такое восстановление потребуется не для всех блоков сетки, то память существенно экономится за счёт частично повторяемых вычислений.

Исследование эффективности алгоритма по используемой памяти может быть проведено аналитически. Оценка используемой памяти алгоритма в байтах содержит два слагаемых: использования памяти на прямом и обратном проходе:

$$3 \times 4 \times 2H \left(\frac{L-1}{H} \right)^2 + 3\frac{2}{8}H^2 \approx 24\frac{L^2}{H} + \frac{3}{4}H^2. \tag{23}$$

Первое слагаемое получается следующим образом: количество блоков сетки $\left(\frac{L-1}{H}\right)^2$ умножается на количество начальных данных для каждого блока (два вектора) – $2H$, которые в свою очередь умножаются на количество байт одной ячейки матрицы – 3×4. Основное значение веса ячейки, кодируемое 4 байтами в одном из форматов int32array или float32array, умножается на 3 по числу компонент вектора весов для реализации аффинной системы штрафов.

На обратном проходе при восстановлении матрицы внутри блока нет необходимости сохранять веса для ячеек, а только вектор вариантов, или даже, более того, номер элементарной редакции после применения оператора выбора приоритета. Значение номера (1, 2 или 3) кодируется 2 битами и может быть упаковано по 4 в один байт. После умножения на коэффициент 3, требуемый для реализации аффинной системы штрафов, получается второй член в формуле (23).

Минимальное значение правой части (аппроксимации) выражения (23) может быть

найдено из необходимого условия экстремума и достигается при:

$$H = 2 \times 2^{1/3} L^{2/3}. \tag{24}$$

Минимальное значение необходимой памяти получается при подстановке (24) в (23):

$$9 \times 2^{2/3} L^{4/3}. \tag{25}$$

Таким образом, формула (24) представляет оптимальный выбор размера сетки для минимизации памяти. Такой выбор является результатом компромисса между использованием оперативной памяти на прямом и обратном проходе. Общий объем памяти в этом случае оценивается формулой (25).

В более общем случае, в котором различаются как длины сравниваемых последовательностей L_Q и L_S, так и размеры блока H_Q и H_S, формула количества памяти имеет вид

$$\frac{12\,(H_Q + H_S)\,(L_Q - 1)(L_S - 1)}{H_Q H_S} + \frac{3 H_Q H_S}{4}. \tag{26}$$

Единственный экстремум достигается в точке

$$H_Q = H_S = 2 \times 2^{1/3} L_Q^{1/3} L_S^{1/3}. \tag{27}$$

Значение количества памяти в точке экстремума

$$9 \times 2^{2/3} L_Q^{2/3} L_S^{2/3}. \tag{28}$$

Полученные формулы являются обобщением формул (24) и (25) на случай прямоугольной матрицы и сетки. Заметим, что в случае прямоугольной матрицы оптимальный размер блока сетки получается квадратным $H_Q = H_S$. Это можно объяснить следующим образом. Поскольку согласно формуле (26) количество используемой памяти пропорционально периметру блока, то наиболее оптимальным является квадратный блок, содержащий максимальное количество элементов при минимальном периметре.

Автоматический выбор параметров алгоритма

Чтобы организовать автоматический выбор параметров параллельного алгоритма, в добавление к приведённым формулам необходимо рассмотреть вырожденный случай, когда матрица может быть с очень большим отношением сторон $L_Q << L_S$. Такая ситуация происходит при поиске наилучшего выравнивания короткой последовательности и некоторой длинной последовательности. В этом случае блок уже не может быть квадратным, и необходимо подбирать размеры блока, исходя из имеющихся ограничений. Для того, чтобы распараллелить алгоритм, необходимо и достаточно, чтобы сетка имела столько строк и столбцов, сколько вычислителей (ядер процессора) M. Для этого размер блока должен удовлетворять неравенствам:

$$H_Q \leqslant \frac{L_Q - 1}{M}, \quad H_S \leqslant \frac{L_S - 1}{M}. \tag{29}$$

С другой стороны, как было показано в предыдущем пункте, площадь блока не может быть меньше некоторого порога. Минимальный размер блока можно установить, исходя из замеров производительности, приведённых в таблице 1, следующим образом:

$$H_Q H_S \geqslant 10^6. \tag{30}$$

На основе формул (27, 29, 30), рассматриваемых в совокупности, можно построить правило выбора величин шагов сетки: сначала шаги сетки устанавливаются равными по формуле (27), затем в случае, если хотя бы одно из неравенств (29) не выполняется, шаги сетки устанавливаются в соответствии со своими предельными значениями, задаваемыми неравенствами (29), наконец, если неравенство (30) не удовлетворяется, то распараллеливание алгоритма теряет смысл, и шаги сетки следует приравнять максимальным значениям $H_Q = L_Q - 1, H_S = L_S - 1$.

ЗАКЛЮЧЕНИЕ

В данной работе реализован алгоритм глобального выравнивания последовательностей, как нуклеотидных, так и аминокислотных. При этом алгоритм обладает возможностью изменения всех параметров редакционного расстояния и выбора приоритета обратного прохода метода. Существенным свойством алгоритма является его способность к выравниванию длинных последовательностей за счёт минимизации используемой памяти и параллельности. Получены точные оценки используемой памяти и оптимального выбора параметров распараллеливания.

Алгоритм реализован в виде веб-приложения, загружаемого с сервера и выполняемого на стороне клиента в виртуальной машине веб-браузера. Алгоритм с выбранными значениями параметров по умолчанию выдает результаты в точности соответствующие алгоритму, используемому в сервисе глобального выравнивания NCBI BLAST [7]. Тестирование разработанного сервиса показало, что эффективность работы приложения сравнима с другими сервисами глобального выравнивания для длин последовательностей порядка 10^4, но в отличие от аналогов разработанный сервис способен выравнивать последовательности порядка 10^6.

Работа выполнена при поддержке РФФИ, проекты №15-29-07063 и №16-01-00692.

СПИСОК ЛИТЕРАТУРЫ

1. Оплачко Е.С., Устинин Д.М., Устинин М.Н. Облачные технологии и их применение в задачах вычислительной биологии. *Математическая биология и биоинформатика.* 2013. Т. 8. № 2. С. 449–466. doi: 10.17537/2013.8.449
2. Daugelaite J., O'Driscoll A., Sleator R. An Overview of Multiple Sequence Alignments and Cloud Computing in Bioinformatics. *ISRN Biomathematics.* 2013. V. 2013. P. 1–14.
3. Camacho C., Coulouris G., Avagyan V., Ma N., Papadopoulos J., Bealer K., Madden T.L. BLAST+: architecture and applications. *BMC Bioinformatics.* 2009. V. 10. P. 421.
4. Needleman S., Wunsch C. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.* 1970. V. 48. № 3. P. 443–453.
5. Панкратов А.Н., Пятков М.И., Тетуев Р.К., Назипова Н.Н., Дедус Ф.Ф. Поиск протяженных повторов в геномах на основе спектрально-аналитического метода. *Математическая биология и биоинформатика.* 2012. Т. 7. № 2. С. 476–492. doi: 10.17537/2012.7.476
6. Pyatkov M.I., Pankratov A.N. SBARS: fast creation of dotplots for DNA sequences on different scales using GA–,GC–content. *Bioinformatics*. 2014. V. 30. № 12. P. 1765–1766.
7. *NCBI BLAST*: web site. URL: https://blast.ncbi.nlm.nih.gov/Blast.cgi (дата обращения: 14.04.2017).
8. Rice P., Longden I., Bleasby A., EMBOSS: The European Molecular Biology Open Software Suite. *Trends in Genetics*. 2000. V. 16. № 6. P. 276–277.

9. Gotoh O. An improved algorithm for matching biological sequences. *J. Mol. Biol.* 1982. V. 162. № 3. P. 705–708.
10. Press W., Teukolsky S., Vetterling W., Flannery B. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 2007. 1256 p.
11. Myers E., Miller W. Optimal alignments in linear space. *Comput. Appl. Biosci.* 1988. V. 4. № 1. P. 11–17.
12. Hirschberg D.S. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*. 1975. V. 18. № 6. P. 341–343.
13. Altschul S., Gish W., Miller W., Myers E., Lipman D. Basic local alignment search tool. *J. Mol. Biol.* 1990. V. 215. P. 403–410.
14. Driga A., Lu P., Schaeffer J., Szafron D., Charter K., Parsons I. FastLSA: A Fast, Linear-Space, Parallel and Sequential Algorithm for Sequence Alignment. *Algorithmica*. 2006. № 45. P. 337–375.
15. Chakraborty A., Bandyopadhyay S. FOGSAA: Fast Optimal Global Sequence Alignment Algorithm. *Scientific Reports*. 2013. № 3. P. 1746.
16. Loving J., Hernandez Y., Benson G. BitPAl: a bit-parallel, general integer-scoring sequence alignment algorithm. *Bioinformatics*. 2014. V. 30. № 22. P. 3166–3173.
17. Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. *Bioinformatics*. 2007. V. 23. № 2. P. 156–161.
18. Huson D., Chao Xie C. A poor man's BLASTX - high-throughput metagenomic protein database search using PAUDA. *Bioinformatics*. 2014. V. 30. № 1. P. 38–39.
19. Galvez S., Diaz D., Hernandez P., Esteban F.J., Caballero J.A., Dorado G. Next-generation bioinformatics: using many-core processor architecture to develop a web service for sequence alignment. *Bioinformatics*. 2010. № 26. P. 683–686.
20. Blom J., Jakobi T., Doppmeier D., Jaenicke S., Kalinowski J., Stoye J., Goesmann A. Exact and complete short-read alignment to microbial genomes using Graphics Processing Unit programming. *Bioinformatics*. 2011. № 27. P. 1351–1358.
21. Левенштейн В.И. Двоичные коды с исправлением выпадений, вставок и замещений символов. *Доклады Академии Наук СССР*. 1965. Т. 163. № 4. С. 845–848.
22. Dayhoff M., Schwartz R., Orcutt B. A model of Evolutionary Change in Proteins. *Atlas of protein sequence and structure.* 1978. V. 5. P. 345–358.
23. Henikoff S., Henikoff G. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA.* 1992. V. 89. № 22. P. 10915–10919.
24. Hamming R.W. Error Detecting and Error Correcting Codes. *The Bell System Technical Journal.* 1950. V. 29. № 2. P. 147–160.
25. Xuhua X. In: *Bioinformatics and the Cell. Modern Computational Approaches in Genomics, Proteomics and Transcriptomics*. Springer, 2007. P. 124–127.
26. Ibarra I., Melo F. Interactive software tool to comprehend the calculation of optimal sequence alignments with dynamic programming. *Bioinformatics*. 2010. V. 26. № 13. P. 1664–1669.

Рукопись поступила в редакцию 14.03.2017
Дата опубликования 13.04.2017