Информационные процессы, Том 2, № 1, стр. 85–90

Параллельный алгоритм поиска регуляторного сигнала в геномах бактерий

С.Н.Истомина, Л.И. Рубанов

Институт проблем передачи информации, Российская академия наук, Москва, Россия
Поступила в редколлегию 17.05.2002

Аннотация—Описывается способ распараллеливания существующего алгоритма поиска регуляторного сигнала. Полученный параллельный алгоритм, использующий стандартный протокол MPI, не привязан к определенному числу процессоров и характеризуется линейной зависимостью скорости вычислений от числа доступных процессоров. С использованием данного алгоритма становится возможным проводить поиск регуляторных сигналов в больших наборах геномных последовательностей, что было продемонстрировано в серии реальных экспериментов с предлагаемым параллельным алгоритмом.

1. ВВЕДЕНИЕ

Актуальная задача информационной генетики – поиск регуляторного сигнала в геномах бактерий – с математической точки зрения формулируется следующим образом. Дан набор из n нуклеотидных последовательностей, длины которых не фиксированы или примерно одинаковы (и в этом случае равны каждая какому-то числу m). *Системой* называется набор слов фиксированной длины l, не более чем по одному слову из каждой последовательности; цель решения задачи состоит в нахождении системы, которая бы состояла из как можно более попарно похожих друг на друга слов (из возможно большего числа последовательностей).

Похожесть понимается, например, в смысле суммы попарных расстояний Хэмминга между словами, входящими в систему, или в смысле какой-то другой фиксированной метрики таких слов, а в более общем виде – в смысле максимизации какого-то фиксированного функционала качества системы. Качество системы определяется, например, как сумма попарных «расстояний» между ее словами, вычисляемых с помощью функции $F(x, y)$, которая для двух слов x и y длины l отражает степень их похожести между собой (например, количество совпадающих в них букв на соответственных местах), а также отражает степень наличия других желательных характеристик у такой пары слов. Интуитивно такая система понимается как *сигнал*; а слово, являющееся представителем сигнала в какой-то из исходных последовательностей, – как *регуляторный участок* (*сайт*) в этой последовательности.

Основой для данной работы послужил алгоритм поиска регуляторного сигнала [1], реализующий поиск квазиоптимального решения путем направленного частичного перебора. При фиксированном значении l этот алгоритм решает исходную задачу за время, квадратичное от числа n исходных последовательностей и кубичное от длины m каждой из них. Несмотря на столь низкую трудоемкость алгоритма по сравнению с полным перебором, который в данной задаче потребовал бы просмотра $(m - l)^n$ вариантов, для практически интересных значений n и m (порядка от сотни до тысячи) время работы алгоритма на доступных однопроцессорных компьютерах оказывается значительным, что и приводит к естественной идее распараллеливания этого алгоритма.

2. ТРУДНОСТИ ПАРАЛЛЕЛЬНОЙ РЕАЛИЗАЦИИ

Параллельную реализацию существующего алгоритма осложняют два фактора: циклическая (т.е. последовательная) структура исходного алгоритма и модель общей памяти, неявно лежащая в его основе и затрудняющая распределение данных в мультипроцессорной вычислительной установке (как известно, в большинстве современных параллельных компьютеров основной объем оперативной памяти более или менее равномерно распределен между процессорами,

а совместно используемая ими общая память имеет небольшую емкость или вообще отсутствует; при этом обмен информацией между процессорами осуществляется посредством передачи коротких сообщений по внутренней коммуникационной сети, например, с использованием протокола MPI [2, 3], что характеризуется значительно большими временными задержками, чем при непосредственном обращении к оперативной памяти). Обсудим теперь более подробно первую из указанных трудностей.

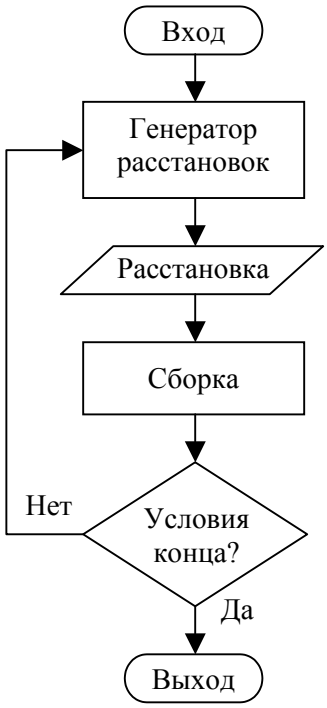

Рис. 1. Базовый алгоритм.

Предложенный ранее последовательный алгоритм имеет следующую укрупненную структуру (рис.1), в которой на каждой итерации ищется квазиоптимальное решение для одного способа нумерации исходных нуклеотидных последовательностей. (Для краткости этот способ нумерации назовем «расстановкой», а процесс решения задачи для данной расстановки – «сборкой» этой расстановки). После сборки очередной расстановки алгоритм генерирует следующую расстановку, и процесс повторяется. Если бы перебрать все $n!$ возможных расстановок (что совершенно недостижимо), то среди всех найденных решений можно выбрать наилучшее (оно и будет оптимальным). Реально же условия окончания цикла в данном алгоритме выбираются так, чтобы получаемое в конце квазиоптимальное решение было близко к оптимальному в смысле применяемого функционала качества, для чего предложены два принципа (и, соответственно, два критерия).

Первый принцип учитывает качество найденного сигнала и сравнительный вклад в него отдельных последовательностей (те из них, которые дают более «ценные» слова системы, в следующей расстановке получают меньшие порядковые номера). Второй принцип учитывает полноту покрытия множества последовательностей основными ребрами графа G, с помощью которого в алгоритме проводится поэтапное дихотомическое разбиение этого множества. Ранее нами были опробованы варианты последовательного алгоритма поиска регуляторного сигнала, в которых реализованы различные схемы совместного использования указанных двух принципов (подробнее см. [1, 4]).

Возвращаясь к задаче распараллеливания этого алгоритма, заметим, что основной объем вычислений (и, значит, время работы алгоритма) связан с процессом сборки. Поскольку этот блок находится внутри последовательного цикла (рис. 1), единственная возможность распараллеливания алгоритма просматривается только в организации параллельных вычислений внутри блока. В принципе, это возможно, однако процесс сборки построен так, что в нем если и есть какая-то пригодная для параллельных расчетов векторная структура данных (в первую очередь это разбиение набора нуклеотидных последовательностей в соответствии с текущим уровнем дерева G), то размерность этой структуры жестко связана с числом n последовательностей в решаемой задаче, и притом непостоянна на разных этапах сборки, по мере продвижения по дереву. К тому же время вычислений для отдельных элементов такого вектора (т.е. пар последовательностей) может меняться в широких пределах, а для перехода на следующий уровень необходимы результаты по всем элементам, так что попытка жесткого закрепления отдельных этапов или однородных элементов сборки за процессорами не позволяет ни эффективно использовать все имеющиеся в конкретной вычислительной установке процессоры, ни даже равномерно загрузить те, которые задействованы. Становится ясно, что эффективное распараллеливание невозможно без существенной переработки общей логики алгоритма.

3. ДВУМЕРНЫЙ СПИСОК РАССТАНОВОК И ВОЛНОВАЯ СХЕМА ВЫЧИСЛЕНИЙ

Если взглянуть на алгоритм [1] как на метод оптимизации заданного функционала качества, то можно усмотреть следующие аналогии. Каждая расстановка представляет собой аналог точ-

ки в пространстве поиска экстремума, а процесс сборки по своему результату аналогичен вычислению значения функционала в этой точке. (Разумеется, это всего лишь аналогия, и мы не пытаемся оценить метрические свойства такого дискретного пространства, насчитывающего в точности $n!$ точек). В то же время видно, что алгоритм использует точки двух видов, генерируемые путем применения, соответственно, первого и второго принципов, упоминавшихся выше. В рамках предлагаемой аналогии их соотношение таково. Второй принцип (максимальной покрытости) порождает новые базовые точки для *глобального* поиска оптимума (что оправдано, поскольку мы не имеем сведений о геометрии поверхности отклика), тогда как с помощью первого принципа мы стремимся достичь наилучшего *локального* решения, стартуя с выбранной базовой точки. Эта двоякая природа генерируемых расстановок, незаметная в последовательном алгоритме рис. 1 (а скорее затрудняющая его реализацию), как будет видно из дальнейшего, и может быть эффективно использована для построения именно параллельного алгоритма поиска регуляторного сигнала.

Общую идею предлагаемого метода распараллеливания можно описать так: все доступные процессоры параллельного вычислительного комплекса (кроме выделенного одного) загружаются локальной оптимизацией, каждый из своей базовой точки. При этом все необходимые для применения первого принципа данные сохраняются в локальной памяти своего процессора, а в выделенную корневую ветвь передаются только результаты вычислений (обнаруженные сигналы вместе с их характеристиками). Освободившиеся после окончания локальной оптимизации (этот процесс назовем «вытягиванием») процессоры получают для обработки новые базовые расстановки, и т.д. Критерием окончания будет исчерпание всего набора базовых точек, подготовленного исходя из второго принципа (покрытости), и завершение вычислений во всех параллельных ветвях.

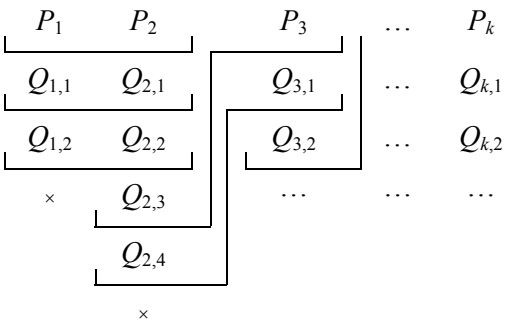

Рис. 2. <*P, Q*>-списки расстановок и порядок их параллельной сборки.

Для реализации этой идеи будем генерировать уже не линейную очередь расстановок, как в последовательном алгоритме, а двумерный список <*P, Q*> (рис. 2). Он образован *P*-списком расстановок P_1, P_2,…, P_k, который строится до начала работы алгоритма; длина k этого списка определяется желаемой степенью покрытости (например, если потребовать, чтобы каждая пара исходных последовательностей хотя бы один раз встречалась на основном ребре верхнего уровня *G*-дерева, т.е. соответствующие последовательности попадали в различные половины при самом первом разбиении всего множества, то, очевидно, величина $k = n(n\text{-}1)/2$). Далее, для каждого элемента *P*-списка строится свой *Q*-список, которому отвечает процесс вытягивания. Каждый такой *Q*-список строится динамически в процессе работы алгоритма: после анализа качества сигналов, уже найденных в *i*-м *Q*-списке (включая и его корень P_i), список либо продолжается, т.е. строится очередная расстановка $Q_{i,j}$, либо прерывается, что на рис. 2 символически обозначено крестиком. В последнем случае процессор, ранее обслуживавший *i*-й *Q*-список, освобождается и принимает в работу очередной необработанный элемент *P*-списка, вытягивая из него новый *Q*-список.

Легко видеть, что в такой двумерной структуре расстановок параллельная обработка организована по типу вычислительной волны, фронт которой распространяется из левого верхнего угла структуры в направлении вниз и вправо, пока не будут обработаны все расстановки <*P, Q*>-списков. На рис. 2 последовательные положения этого волнового фронта для наглядности показаны для случая двух процессоров и в предположении постоянной длительности сборки для всех расстановок, однако легко представить себе и работу такого алгоритма в общем случае. Для примера на рис. 3 приводится начало хода решения на 8 процессорах (не считая обслуживающего корневую ветвь) одной из реальных задач с размерностями n=45, m=201, l=15. Поскольку изобразить движение фронта вычислений на статическом рисунке затруднительно, в скобках просто указана очередность, в которых расстановки передавались свободным процессорам. Под каждой расстановкой записано значение качества сигнала, найденного в результате

ее сборки; выделены максимальные значения в пределах каждого Q-списка, подчеркнуто текущее максимальное значение (все это на момент, зафиксированный на рисунке).

$P_0(0)\rightarrow$ 7734	$P_1(1)\rightarrow$ 9064	$P_2(2)\rightarrow$ 8872	$P_3(3)\rightarrow$ **11468**	$P_4(4)\rightarrow$ 9314	$P_5(5)\rightarrow$ 8746	$P_6(6)\rightarrow$ 8696	$P_7(7)\rightarrow$ 9138	$P_8(59)\rightarrow$ **9622**	$P_9(68)\rightarrow\ldots$ 8106
$\downarrow Q_{0,0}(11)$ 9972	$\downarrow Q_{1,0}(12)$ 10654	$\downarrow Q_{2,0}(14)$ 10776	$\downarrow Q_{3,0}(13)$ 9310	$\downarrow Q_{4,0}(8)$ 8348	$\downarrow Q_{5,0}(10)$ 12150	$\downarrow Q_{6,0}(9)$ 11756	$\downarrow Q_{7,0}(15)$ **10018**	$\downarrow Q_{8,0}(66)$ 9140	$\downarrow Q_{9,0}(75)$ **11472**
$\downarrow Q_{0,1}(18)$ 7860	$\downarrow Q_{1,1}(19)$ **11006**	$\downarrow Q_{2,1}(22)$ 9348	$\downarrow Q_{3,1}(21)$ 9584	$\downarrow Q_{4,1}(17)$ 9388	$\downarrow Q_{5,1}(16)$ 12056	$\downarrow Q_{6,1}(20)$ **12628**	$\downarrow Q_{7,1}(23)$ 8408	$\downarrow Q_{8,1}(76)$ 8996	. . .
$\downarrow Q_{0,2}(26)$ 8372	$\downarrow Q_{1,2}(27)$ 10522	$\downarrow Q_{2,2}(31)$ 8612	$\downarrow Q_{3,2}(28)$ 8922	$\downarrow Q_{4,2}(25)$ 8602	$\downarrow Q_{5,2}(24)$ 12040	$\downarrow Q_{6,2}(30)$ 11200	$\downarrow Q_{7,2}(29)$ 8996	. . .	
$\downarrow Q_{0,3}(34)$ 9784	$\downarrow Q_{1,3}(38)$ 8394	$\downarrow Q_{2,3}(39)$ 11082	$\downarrow Q_{3,3}(35)$ 8734	$\downarrow Q_{4,3}(33)$ 9022	$\downarrow Q_{5,3}(32)$ 11942	$\downarrow Q_{6,3}(36)$ 11626	$\downarrow Q_{7,3}(37)$ 9416		
$\downarrow Q_{0,4}(42)$ 10188	$\downarrow Q_{1,4}(44)$ 8716	$\downarrow Q_{2,4}(47)$ 9928	$\downarrow Q_{3,4}(43)$ 9238	$\downarrow Q_{4,4}(41)$ 9284	$\downarrow Q_{5,4}(40)$ 11448	$\downarrow Q_{6,4}(45)$ 9262	$\downarrow Q_{7,4}(46)$ 8744		
$\downarrow Q_{0,5}(50)$ 10100	$\downarrow Q_{1,5}(52)$ 9546	$\downarrow Q_{2,5}(54)$ 9564	$\downarrow Q_{3,5}(51)$ 8958	$\downarrow Q_{4,5}(49)$ **9490**	$\downarrow Q_{5,5}(48)$ 12164	$\downarrow Q_{6,5}(53)$ 11526	$\downarrow Q_{7,5}(55)$ 8448		
$\downarrow Q_{0,6}(58)$ **12850**	$\downarrow Q_{1,6}(60)$ 10982	$\downarrow Q_{2,6}(63)$ 11702	×	$\downarrow Q_{4,6}(57)$ 8634	$\downarrow Q_{5,6}(56)$ **12668**	$\downarrow Q_{6,6}(61)$ 12104	$\downarrow Q_{7,6}(62)$ 8358		
$\downarrow Q_{0,7}(67)$ 12054	×	$\downarrow Q_{2,7}(70)$ **11766**		$\downarrow Q_{4,7}(65)$ 9088	$\downarrow Q_{5,7}(64)$ 12286	$\downarrow Q_{6,7}(71)$ 12058	$\downarrow Q_{7,7}(69)$ 8714		
$\downarrow Q_{0,8}(74)$ 8918		$\downarrow Q_{2,8}(78)$ 10306		$\downarrow Q_{4,8}(73)$ 8112	$\downarrow Q_{5,8}(72)$ 11632	×	$\downarrow Q_{7,8}(77)$ 8492		
. . .		. . .		. . .	. . .		. . .		

Рис. 3. Ход решения задачи на 8 процессорах (n=45, m=201, l=15).

С точки зрения эффективности распараллеливания описанная схема обладает многими преимуществами, из которых отметим три наиболее существенных:

1. Данная вычислительная схема позволяет сразу загрузить большое число процессоров (по меньшей мере столько, какова длина P-списка k).

2. Все вторичные вычислительные ветви работают по одному и тому же алгоритму, а именно осуществляют сборку для поданной на вход расстановки, и этот алгоритм может выполняться в каждой такой ветви независимо от других.

3. Информационный обмен между каждой из вторичных ветвей и корневой ветвью алгоритма характеризуется небольшой интенсивностью: в сторону вторичной ветви передается только расстановка (n целочисленных значений), а в сторону корневой – найденный сигнал и значения качества каждого из его слов ($2n$ целочисленных значений).

Указанные преимущества позволили построить параллельный алгоритм, не привязанный к какому-то конкретному вычислительному комплексу, а ориентированный на широкий класс параллельных систем типа кластеров и суперкомпьютеров, в которых имеется эффективная программно-аппаратная реализация по меньшей мере первой редакции протокола MPI [2].

Необходимости использования общей памяти удалось избежать благодаря переносу всего процесса проверки критериев и генерации расстановок в корневую ветвь алгоритма, где собираются результаты сборки от всех вторичных ветвей. Единственный общий информационный массив (матрица попарных близостей всевозможных слов данной длины из всех исходных последовательностей) не меняется в процессе работы алгоритма, что позволяет независимо вычислить и сохранить его в каждой из вторичных ветвей до начала работы собственно алгоритма поиска оптимального сигнала (заметим, что корневая ветвь в это время все равно занята построением P-списка с заданной степенью покрытости).

3. ПОДРОБНОСТИ РЕАЛИЗАЦИИ И РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

Описанный выше параллельный алгоритм поиска регуляторного сигнала реализован в виде 32-разрядного консольного приложения, написанного на языке С в стандарте ANSI. Без каких-

либо изменений программа компилируется трансляторами Borland C++ 5.02, gcc, pgcc и работает под управлением операционных систем Windows9x/NT/ME/2000 или Linux.

Заметим, что с целью экономии дорогостоящего машинного времени параллельных вычислительных комплексов отладка программы и первые тестовые просчеты проводились на PC в среде Windows98/2000. Однако, в отличие от последовательного алгоритма, переносимость которого на параллельную вычислительную систему обеспечивается лишь соблюдением при написании программы стандарта языка ANSI C, для написания и отладки параллельного алгоритма необходимо было организовать на одном или нескольких PC действующую среду интерфейса MPI в версии, совместимой с используемой на кластере.

После ряда экспериментов мы остановились на коммерческом программном продукте WMPI версии 1.54 компании Critical Software, на использование пробной версии которого от разработчика была получена 120-дневная лицензия. Несмотря на то, что продукт рассчитан на программирование в среде Microsoft Visual C 6.0, нам удалось привязать входящие в состав библиотеки к использовавшемуся компилятору Borland C++ 5.02. Это позволило полностью провести отладку и контрольные просчеты с имитацией различного числа ветвей на одном персональном компьютере. В случае необходимости программа полностью готова к использованию для решения реальных задач на распределенном вычислительном комплексе в виде группы PC, связанных сетью протокола TCP/IP.

Такой подход полностью оправдал себя, позволив ускорить сроки разработки и сократить затраты на ее проведение. После окончания отладки параллельная программа была перенесена сразу на ряд вычислительных систем и не потребовала дополнительной отладки.

По результатам проведенных расчетов на модельных и реальных примерах была доказана высокая степень эффективности распараллеливания алгоритма: коэффициент полезного использования процессорного времени во вторичных ветвях по данным профилирования с помощью программы Vampir составил 94-96%. Кроме того, эксперименты подтвердили ранее предсказанные теоретически линейные зависимости времени сборки от числа исходных последовательностей (рис. 4) и быстродействия алгоритма от количества доступных процессоров (рис. 5). При каком числе процессоров линейный рост производительности будет замедляться, установить с помощью доступных вычислительных комплексов оказалось невозможным.

В заключение заметим, что в настоящее время с помощью описанного параллельного алгоритма проводятся реальные расчеты на 768-процессорной суперЭВМ МВС-1000М Межведомственного суперкомпьютерного центра РАН, РФФИ, Минпромнауки и Минобразования РФ, которая имеет пиковую производительность около 1 Тфлопс.

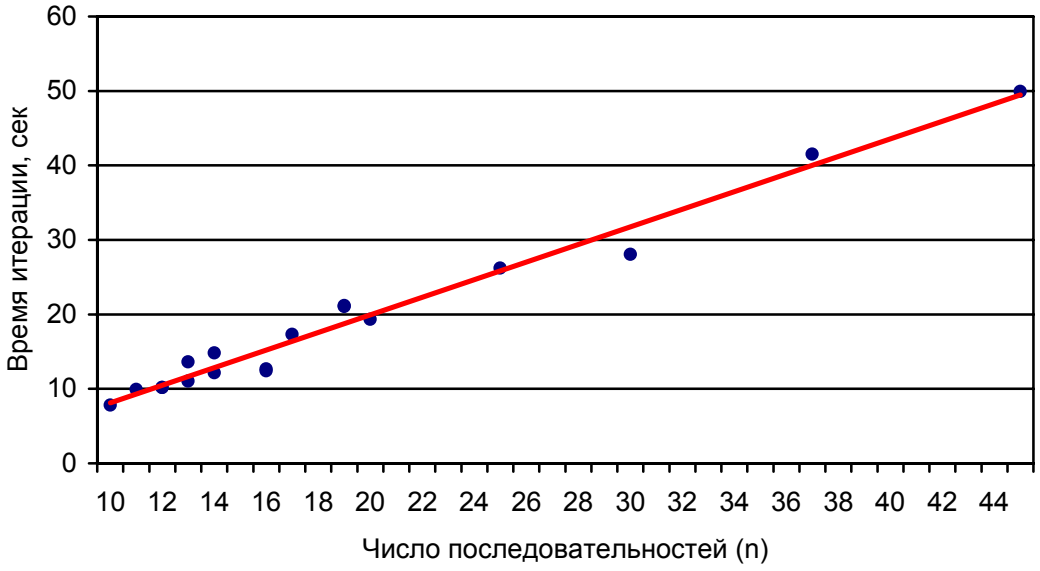

Рис. 4. Зависимость времени сборки от размерности задачи.

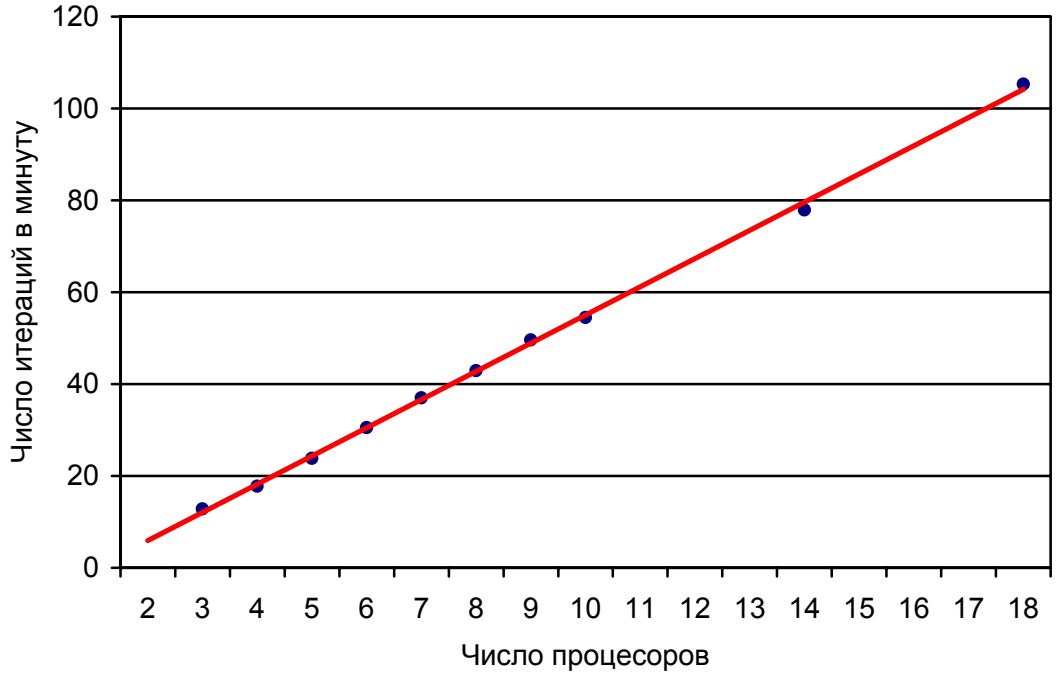

Рис. 5. Рост производительности параллельного алгоритма (n=14, m=200, l=20).

СПИСОК ЛИТЕРАТУРЫ

1. Данилова Л.В., Горбунов К.Ю., Гельфанд М.С., Любецкий В.А. Алгоритм выделения регуляторных сигналов в последовательностях ДНК (2). *Молекулярная биология*, 2001, том 35, № 6, стр. 987–995.
2. MPI: A Message-Passing Interface Standard. *Message Passing Interface Forum*, Version 1.1: June 1995. Knoxville, University of Tennessee, 1995.
3. MPI-2: Extensions to the Message-Passing Interface. *Message Passing Interface Forum*, July 18, 1997. Knoxville, University of Tennessee, 1997.
4. Данилова Л.В., Любецкий В.А. Алгоритм выделения регуляторных сигналов: тестирование и биологические применения. В кн. *Труды 3-й международной конференции «Проблемы управления и моделирования в сложных системах»*, Самара, РАН, 2001, стр. 632–634.

Статью представил к публикации член редколлегии В.А.Любецкий