УДК 577.322.23; 004.272.2

РЕАЛИЗАЦИЯ ПОДДЕРЖКИ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ В ПРОГРАММАХ ДОКИНГА SOLGRID И SOL

И. В. Офёркин[1], А. В. Сулимов[1], О. А. Кондакова[1], В. Б. Сулимов[1]

Рассмотрены схемы реализации и получившиеся эффективности для нескольких вариантов докинга одного лиганда с использованием параллельных вычислений. В качестве программ, для которых реализовывались параллельные версии, были выбраны программы SOLGRID и SOL. Тестирование проводилось на кластерном суперкомпьютере СКИФ МГУ "Чебышёв". Для реализации параллельности вычислений использовался MPI. Данная работа выполнена в рамках проведения научно-исследовательских работ по пост-геномным исследованиям и технологиям МГУ им. М. В. Ломоносова и выполнения работ по госконтракту № 02.740.11.0388 по теме "Суперкомпьютерные технологии для решения задач обработки, хранения, передачи и защиты информации", а также частично поддержана грантами РФФИ (коды проектов 09–01–12097_офим, 10–07–00595-а).

Ключевые слова: докинг, высокопроизводительные вычисления, интерфейс передачи сообщений.

1. Введение. Главная парадигма [1], используемая при современной разработке новых лекарств, заключается в следующем. Многие болезни связаны с функционированием определенных белков, поэтому для лечения заболеваний надо блокировать работу этих белков. Например, белок-мишень может принадлежать вирусу, и его блокирование позволяет сделать невозможным размножение вируса в организме. Блокирование осуществляется с помощью молекул, которые избирательно связываются с этими белками в организме. Такие молекулы, составляющие основу новых лекарств, называются ингибиторами. Как правило, ингибиторы — это сравнительно небольшие органические молекулы, связывающиеся с определенными областями белков-мишеней, называемыми центрами связывания или активными центрами. Поиск молекул-ингибиторов для заданного белка-мишени составляет начальный этап разработки нового лекарства. Быстрое и эффективное решение этой задачи в значительной степени определяет минимизацию материальных затрат и продолжительность последующих этапов разработки нового лекарства. По времени этап разработки новых ингибиторов занимает примерно 50% от общей длительности разработки нового лекарства.

Существенно сократить затраты времени и средств на этапе поиска ингибиторов можно с помощью методов компьютерного молекулярного моделирования, среди которых главную роль играет докинг. Докинг — это позиционирование молекул-кандидатов в ингибиторы в активном центре белка-мишени и оценка их энергии связывания. Чем сильнее молекула связывается с белком, тем лучше ингибитор и эффективнее новое лекарство на его основе. В последнее время молекулярное моделирование и, в частности, докинг начинают все более широко применяться для разработки новых лекарственных средств [1–3] наряду с традиционной методологией экспериментального перебора путем высокопроизводительного скрининга (HTS, High Throughput Screening).

С математической точки зрения задача докинга состоит из трех подзадач:

1) необходимо задать пространство всех возможных способов связывания лиганда с белком-мишенью, то есть определить, какими степенями свободы будут наделены лиганд и белок-мишень;

2) необходимо построить эффективный алгоритм поиска в этом многомерном пространстве наилучшего способа связывания лиганда с белком-мишенью;

3) необходимо составить скоринг-функцию, которая определяет энергию связывания лиганда с белком-мишенью; эта функция и служит объектом глобальной оптимизации для алгоритма поиска.

Пространство всех возможных способов связывания лиганда с белком-мишенью определяет, какие изменения в положениях лиганда и белка-мишени считаются возможными при образовании комплекса.

[1] Научно-исследовательский вычислительный центр, Московский государственный университет им. М. В. Ломоносова, 119991, Ленинские горы, Москва; ООО "Димонта", ул. Нагорная, д. 15, корп. 8, 117186, Москва; И. В. Офёркин, программист, e-mail: io@dimonta.com; А. В. Сулимов, системный программист, e-mail: sulimovv@mail.ru; О. А. Кондакова, ст. науч. сотр., e-mail: olga.kondakova@srcc.msu.ru; В. Б. Сулимов, зав. лабораторией, e-mail: vladimir.sulimov@gmail.com

Чем больше рассматривается таких возможных изменений, тем выше размерность этого пространства, следовательно, тем сложнее будет осуществлять поиск в этом пространстве. Однако чем меньше допускается таких изменений, тем грубее становится модель докинга, так как при образовании комплекса и лиганд, и белок-мишень могут претерпевать значительные конформационные изменения [4–7]. Самая простая модель связывания лиганда с белком-мишенью — это модель жесткого лиганда и жесткого белка-мишени [8]. Такая модель сама по себе применяется сейчас только для белок-белкового докинга [9, 10]. Частичный учет гибкости лиганда и белка-мишени в этом случае осуществляется при помощи "размывания" скоринг-функции, т.е. расширения потенциальных ям межатомных взаимодействий. Более сложная модель докинга — это докинг гибкого лиганда в жесткий белок-мишень. Подвижность лиганда в распространенных программах докинга (AutoDock [11, 12], DOCK [13], ICM [14], GOLD [15, 16]) реализуется за счет изменения его торсионных углов. Такой же способ задания подвижности лиганда используется в программе SOL [17], разработанной в НИВЦ МГУ и ООО "Димонта" и являющейся основным объектом исследования в данной статье. Идея учета гибкости аминокислотных остатков и основной цепи белков-мишеней при проведении процедуры докинга развивается уже достаточно давно, однако только в настоящее время появились модели, которые можно использовать в программах докинга и которые не приводят к катастрофическому замедлению их работы [2]. На сегодняшний день в разных программах докинга учет подвижности белка реализуется при помощи:

— задания вращающихся молекулярных групп в аминокислотных остатках протеина (ICM, AutoDock 4.0);

— подстройки водородов белка-мишени для оптимизации водородных связей (GOLD);

— использования ансамбля конформеров для белка-мишени и докинг во все эти конформеры (IFREDA [18]);

—а также использования "размытого" белка-мишени с учетом его конформационной подвижности (FlexE [19, 20]).

Более полный перечень подходов к описанию подвижности белка-мишени приведен в [21–24]. При виртуальном скрининге (докинге больших баз данных лигандов) белок, как правило, считают жестким.

Алгоритмы поиска наилучшего способа связывания лиганда могут быть разделены на следующие классы: алгоритмы систематического поиска, генетические алгоритмы, поиск методом Монте-Карло, методы молекулярной динамики.

Алгоритм систематического поиска подразумевает полный перебор пространства степеней свободы или некоторой его наиболее интересной части, выбранной из априорных соображений. Для докинга "жесткого" лиганда в "жесткий" белок-мишень можно использовать алгоритм на основе быстрого преобразования Фурье (FFT) [9, 10]. Например, в программе FLOG [25] первоначально создается набор различных конформеров лиганда, каждый из которых затем рассматривается как "жесткое" тело и подвергается процедуре докинга. В программе Glide [26–28] осуществляется полный проход по всему пространству торсионных степеней свободы лиганда, но многие конформеры не рассматриваются, исходя из стереохимических соображений. Также глобальный оптимум скоринг-функции может находиться поэтапно, где на каждом этапе изменяется лишь небольшая группа степеней свободы, в то время как остальные переменные полагаются фиксированными. С точки зрения физики это означает, что процедуре докинга подвергаются небольшие части лиганда, которые затем собираются в целый лиганд. Например, в программе FlexX [19, 20] лиганд разбивается на "жесткое" ядро и заместители, которые ранжируются по мере удаления от ядра. Соответственно сначала производится докинг ядра, к которому присоединяются заместители по одному. В программе DOCK лиганд разбивается на "жесткие" фрагменты, каждый из которых независимо подвергается процедуре докинга, а затем фрагменты в найденных положениях соединяются обратно в исходный лиганд.

Генетические алгоритмы [29–31] представляют лиганд, находящийся в активном центре, как особь с некоторым генотипом и моделируют процессы эволюции, которые могут происходить в популяции таких особей. При этом эволюция может быть не дарвиновской. Например, особь может улучшать свой генотип, приспосабливаясь к текущим условиям (ламаркианская эволюция). Генетический подход используется в программах AutoDock, GOLD, MolDock [32]. Также генетический алгоритм используется в рассматриваемой программе SOL.

Поиск методом Монте-Карло реализован в программах ICM [14], IFREDA [18], QXP [33]. Основная идея метода Монте-Карло — случайное изменение положений лиганда и анализ изменений энергии связывания лиганда. На основании изменения энергии осуществляется, например, выбор предпочтительного направления случайных изменений или принятие решения об отказе/принятии изменения. Одна из разновидностей метода Монте-Карло реализована, например, в программах pso@autodock [34] и SODOCK [35].

В этой реализации учитываются не только энергии для данного положения лиганда, но и энергии других, одновременно рассматриваемых, положений лиганда.

Воспроизведение эволюции системы лиганд–белок при помощи молекулярной динамики считается самым реалистичным методом из существующих, однако это также самый медленный метод, и обычно его применяют для уточнения решений, найденных другими способами. Один из способов увеличить быстродействие молекулярной динамики — это задание больших, нефизичных, температур системы. Такой способ называется "высокотемпературный отжиг" [36] и может использоваться в программе AutoDock, хотя и не является ее основным методом.

Более подробно существующие алгоритмы докинга описаны в [22, 24, 37, 38].

Скоринг-функция используется для оценки устойчивости комплекса лиганда и белка-мишени. Эта функция должна вычисляться весьма быстро, так как в процессе докинга она будет вычислена тысячи и миллионы раз. Обычно в скоринг-функцию входят следующие слагаемые: энергия Ван-дер-Ваальса, электростатическая энергия, энергия десольватации, энергия внутренних напряжений (деформации валентных связей), энергия водородных связей, энергия взаимодействия ароматических групп.

Конкретные скоринг-функции могут быть построены на основе силовых полей или на основе анализа частот встречаемости атомных контактов в существующих комплексах. Также осуществляется подгонка коэффициентов перед вкладами слагаемых в скоринг-функцию на основе экспериментального материала.

В одной программе может быть заложено несколько скоринг-функций, и при подсчете энергии комплекса белок–лиганд происходит усреднение по значениям нескольких скоринг-функций для этого комплекса [39, 40].

Если белок-мишень полагается жестким, то его воздействие на лиганд может быть заранее просчитано в виде сетки потенциалов, покрывающей активный центр. Такой подход позволяет вычислять взаимодействие лиганда с белком-мишенью значительно быстрее, чем непосредственным способом. Сетка потенциалов используется в программах AutoDock, FlexX, FlexE, ICM, DOCK. Также сетка потенциалов используется в рассматриваемой программе SOL, она вычисляется при помощи программы SOLGRID [17].

Степени свободы для лиганда и белка-мишени, а также скоринг-функция, могут меняться в процессе докинга. В первом приближении может быть использована простая скоринг-функция с небольшим числом степеней свободы у лиганда и белка-мишени, затем, для уточнения результатов предыдущего приближения, скоринг-функция может быть усложнена и могут быть добавлены новые степени свободы для лиганда и белка-мишени. Например, можно использовать такой двухэтапный докинг: на первом этапе проводится глобальный поиск положений лиганда в активном центре, на втором этапе найденные положения уточняются при локальной оптимизации (ICM, QXP, DOCK 4.0).

Для поиска ингибитора к заданному белку можно осуществлять перебор баз данных готовых лигандов, но также существует подход, заключающийся в построении нового лиганда, исходя из свойств активного центра белка [41]. Такой алгоритм реализован, например, в программе ADAPT [42]. Дальнейшее рассмотрение такого алгоритма конструирования ингибитора de novo выходит за рамки данной статьи.

Так как основное предназначение программ докинга — это виртуальный скрининг больших баз химических соединений, то существуют решения, позволяющие проводить этот скрининг на суперкомпьютерных системах. Программы GOLD, FlexX & FlexE используют PVM (Parallel Virtual Machine) для организации одновременного докинга многих лигандов на UNIX-кластерах. В данном случае параллельные вычисления основаны на схеме раздачи заданий узлам и сбора результатов, где одно (неделимое) задание — это докинг одного лиганда. Программа ICM использует для аналогичных целей PBS (Portable Batch System) для запуска на UNIX-кластерах. Программа DOCK использует MPI для запуска на UNIX-кластерах, поддерживает библиотеки MPICH и MPICH2 [43]. В этом случае распределение вычислений снова основано на master-slave схеме, где master-процесс занимается только вводом-выводом и распределением заданий среди slave-процессов. Докинг одного лиганда снова является неделимым заданием для одного процесса. Балансировка нагрузки осуществляется при помощи сортировки базы лигандов по числу атомов и числу торсионных степеней свободы. Также с использованием программы DOCK может быть запущен массовый скрининг по грид-технологии [44]. Для менее известных программ докинга (Glide, Surflex, MolDock, IFREDA, QXP) нет общедоступных средств для организации параллельного скрининга.

Остановимся подробнее на использовании компьютерных ресурсов при запуске программы докинга AutoDock, поскольку она является наиболее распространенным и достаточно популярным инструментом молекулярного докинга [45]. Для виртуального скрининга библиотек авторы программы AutoDock предлагают воспользоваться возможностями различных грид-систем, таких, например, как FightAIDS@Home [46] и Discover Dengue Drugs–Together [47]. Для облегчения процесса виртуального скрининга больших

баз данных с помощью данной программы были созданы специальные надстройки для кластеров, работающих под операционными системами Linux (система DOVIS [48, 49], GriDock [50] и прочие системы [51]) и Windows (система VSDocker) [52]. Основной целью создания данных систем является максимальная автоматизация процесса докинга, включая подготовку исходных данных (лигандов и белков) и сортировку полученных результатов. Одно неделимое задание в данном случае — это докинг одного или нескольких лигандов на одном узле. Эти системы могут быть интегрированы в существующие грид-системы, а также могут быть установлены на кластерах с различной системой очередей. Параллелизация самой программы AutoDock, а именно генетического алгоритма, в данном случае не проводилась, а в системы встраивалась программа докинга в виде исполняемого модуля.

Программа AutoDock Vina может использовать возможности многоядерных процессоров, запуская одновременную оптимизацию разных положений лигандов [53, 54].

Параллельное выполнение докинга одного лиганда при помощи AutoDock было реализовано в [55]. Для использованных в статье количеств процессоров (1–96) было получено ускорение работы AutoDock, близкое к линейному. Параллельность работы AutoDock была реализована на уровне выполнения независимых запусков ламаркианских генетических алгоритмов на разных узлах кластера. Программа построения сетки потенциалов AutoGrid распараллелена не была.

Также существует реализация алгоритма AutoDock для выполнения на графических картах с использованием технологии CUDA [56, 57]. Практически значимых результатов пока не достигнуто, проекты находятся в стадии разработки.

Характерная продолжительность построения сетки потенциалов при помощи SOLGRID на одном ядре типового компьютера серверного класса составляет 1–3 часа, докинг одного лиганда при помощи программы SOL в аналогичных условиях занимает около 1–6 часов. Для докинга одного лиганда это слишком большие интервалы времени, особенно когда параметры следующего запуска докинга определяются результатами предыдущего докинга. В то же самое время сегодня рядовому пользователю вполне доступны кластерные системы из сотен вычислительных узлов, поэтому становится актуальной возможность осуществлять построение сетки потенциалов и докинг одного лиганда на суперкомпьютерах. В данной работе предлагается вариант организации параллельной работы программ SOL и SOLGRID на основе MPI для докинга одного лиганда.

2. Схема докинга. Для решения задачи докинга требуется отыскать наиболее выгодное положение лиганда в активном центре белка, в котором энергия связывания лиганда с белком будет максимальной. Для упрощения поиска этого наилучшего положения в программе SOL были приняты следующие приближения:

1) лиганд может изменять свое положение при помощи торсионных вращений вокруг одинарных связей,

2) длины связей и валентные углы между связями лиганда в процессе докинга остаются неизменными,

3) поиск положений лиганда осуществляется в некоторой кубической области пространства, содержащей активный центр белка,

4) белок предполагается жестким,

5) согласно формализму силового поля MMFF94 [58] вычисляется энергия взаимодействия лиганда с белком,

6) влияние растворителя учитывается при помощи обобщенной модели Борна [59, 60].

Таким образом, задача докинга представляет собой задачу отыскания глобального минимума функции энергии, зависящей от торсионных углов лиганда и его положения в пространстве как целого.

Взаимодействие лиганда с белком является невалентным, поэтому можно заранее просчитать с некоторым шагом в пространстве взаимодействие каждого возможного типа атома лиганда со всем белком. Сетки потенциалов, заменяющие явное задание атомов белка, рассчитывает программа SOLGRID. Полученный файл с сетками потенциалов далее используется в программе SOL при расчетах энергии взаимодействия лиганда с белком.

2.1. Алгоритм работы программы SOLGRID. Программа SOLGRID служит для расчета потенциалов взаимодействий каждого типа атома лиганда во всевозможных положениях, заданных в узлах равномерной кубической сетки, с белком. Полученные таблицы значений потенциалов используются в программе SOL. Область пространства, покрываемая сеткой, выбирается так, чтобы вместить активный центр белка. Обычно шаг этой сетки равен 0.22 $\mathring{\text{A}}$, число узлов — 101 по каждому измерению. В каждом узле сетки вычисляются потенциалы электростатического, вандерваальсового и десольватационного взаимодействия всех атомов белка с каждым возможным типом атома лиганда. Таким образом, получаются трех- и четырехмерные массивы значений, которые записываются в выходной файл с сеткой потенциалов.

Этот файл затем используется в программе SOL для расчета энергии взаимодействия лиганда с белком. Явное задание белка для программы SOL уже не требуется.

2.2. Алгоритм работы программы SOL. Энергия взаимодействия лиганда с белком рассчитывается с использованием сеток потенциалов, создаваемых программой SOLGRID. При этом используется восьмиточечная линейная интерполяция для определения потенциалов в точках между узлами сетки.

Для поиска энергетически наилучшего положения лиганда используется генетический алгоритм (ГА), схема которого приведена на рис. 1. Генетический алгоритм является моделированием процессов эволюции применительно к возможным положениям лиганда. Этот алгоритм является итерационным и вероятностным.

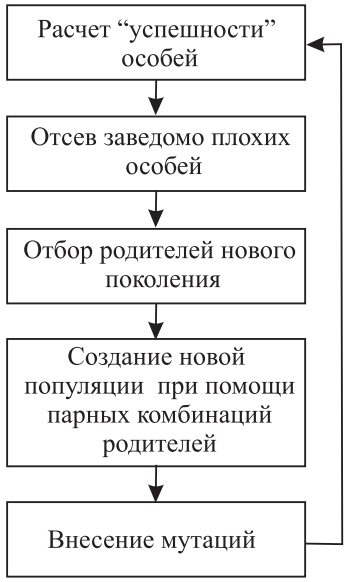

Рис. 1. Схема генетического алгоритма в программе SOL

В терминах генетического алгоритма одно положение лиганда называется особью. Минимальный набор чисел, требуемый для задания особи, называется ее генотипом. В программе SOL генотип особи состоит из координат геометрического центра лиганда, вращений лиганда как целого и углов вращения частей лиганда вокруг одинарных валентных связей.

По заданному генотипу можно рассчитать фенотип особи, т.е. декартовы координаты всех атомов лиганда. Полученный фенотип используется для подсчета энергии комплекса лиганда с белком. Эта энергия служит критерием “успешности” данной особи. Чем она меньше, тем особь “успешнее”.

На каждой итерации генетического алгоритма обрабатывается сразу большой набор особей, который называется популяцией. Номер итерации также называется номером поколения. Для построения популяции следующего поколения из популяции текущего поколения на основании “успешности” выбирается некоторое количество особей, которые называются родителями следующего поколения. Генотипы этих родителей определяют генотипы особей следующего поколения. При отборе родителей учитывается не только “успешность” особей, но и их сходство по генотипу с уже отобранными родителями. Это направлено против “вырождения” популяции, когда все особи задают примерно одно и то же положение лиганда. Небольшое количество самых “успешных” особей переносятся в следующее поколение без изменения. Эта мера направлена против потери достигнутых результатов. Другие же особи следующего поколения получаются при помощи комбинации генов родителей и случайного изменения генов родителей.

От поколения к поколению “успешность” лучших особей монотонно увеличивается. Самая лучшая особь самого последнего поколения является результатом работы генетического алгоритма.

Так как ГА не может гарантировать нахождение глобально лучшего положения лиганда, то для контроля достоверности полученных результатов проводится несколько независимых запусков ГА (рис. 2). Результаты каждого запуска запоминаются, и после проведения всех независимых запусков ГА определяется, насколько похожи их результаты между собой. При достаточно близких результатах, полученных в разных независимых запусках ГА, они считаются достоверными.

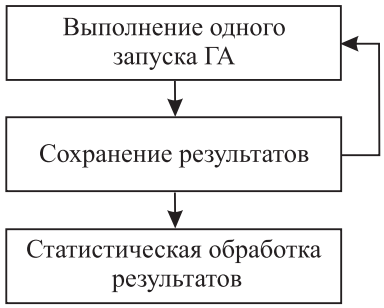

Рис. 2. Схема нескольких запусков генетического алгоритма в последовательной программе SOL

3. Принципы устройства MPI-программ и кластерных систем. Для реализации параллельной работы программ SOLGRID и SOL был выбран MPI (Message Passing Interface). MPI-программа представляет собой параллельно работающие процессы, каждый из которых является полной копией программы по коду и обладает доступом к своей оперативной памяти, недоступной другим процессам. Синхронизация выполнения процессов происходит при помощи явного вызова специальных MPI-функций.

Реализация параллелизма при помощи MPI была выбрана потому, что это распространенная модель параллельного программирования для многопроцессорных систем с распределенной памятью, а самые высокопроизводительные и легкодоступные суперкомпьютеры являются такими системами (кластерные суперкомпьютеры).

3.1. Использованный при разработке и отладке программ суперкомпьютер. Разработка,

отладка и замеры производительности программы проводились на суперкомпьютере СКИФ МГУ "Чебышёв". Основной используемой на суперкомпьютере моделью параллельного программирования является MPI. Суперкомпьютер состоит из 625 вычислительных узлов, каждый узел — это два четырехъядерных процессора Intel Xeon E5472 3.0 ГГц, имеющих общую оперативную память объемом 8 Гб. Вычислительные узлы могут обмениваться данными при помощи MPI-сообщений по коммуникационной сети InfiniBand DDR (Mellanox ConnectX), при этом латентность (задержка между началом передачи MPI-сообщения и началом его приема) на уровне MPI составляет 1.3–1.95 мкс, скорость обмена на уровне MPI составляет 1540 Мб/с. Для ядер, находящихся на одном узле, обмен данными также может осуществляться при помощи вызова MPI-функций. В этом случае MPI-функции осуществляют непосредственное копирование данных из одной области оперативной памяти в другую. В программах SOLGRID и SOL не была задействована возможность явного использования совместного доступа к общей оперативной памяти ядер одного вычислительного узла, так как это бы означало выход за парадигму программирования MPI.

3.2. Показатели качества организации параллельного выполнения MPI-программы. В качестве показателей качества параллельного алгоритма будут использоваться две характеристики: ускорение и эффективность. Пусть программа выполняется на N ядрах, из которых NW ядер занимаются вычислениями, требуемыми для решения задачи, а остальные $NM = (N - NW)$ ядер занимаются координацией работы. Обычно $NM = 0$ или $NM = 1$. Под ускорением $a(N)$ будем понимать уменьшение времени счета $t(N)$ программы на N ядрах по сравнению со временем счета программы при $NW = 1$, то есть когда число ядер, непосредственно занимающихся "полезными" вычислениями, равно одному. Иными словами, $a(N) = \dfrac{t(NW = 1)}{t(N)}$. Под эффективностью $e(N)$ будем понимать долю времени "полезных" вычислений в ядрах, занимающихся "полезными" вычислениями. Эффективность определяется как $e(N) = \dfrac{a(N)}{NW}$. Относительная доля времени простоя ядер, связанная с их синхронизацией и прочими накладными расходами, равна $\big(1 - e(N)\big)$. В идеальном случае, крайне редко достижимом на практике, $a(N) = NW$ и $e(N) = 1$. В реальных задачах $a(N) < NW$ и $e(N) < 1$.

4. Организация параллельного выполнения программы SOLGRID. Потенциалы в каждой точке пространства вычисляются независимо от потенциалов в других областях пространства. Всего существует $101 \times 101 \times 101 = 1030301$ точек пространства, что обеспечивает прекрасную возможность для параллельного вычисления потенциалов в разных точках пространства.

Один из процессов является управляющим и не принимает участия в вычислении потенциалов. Схема его работы приведена на рис. 3а. Он распределяет между остальными подчиненными процессами области пространства (номера элементов в таблицах), в которых необходимо рассчитывать потенциал. Распределение областей пространства осуществляется небольшими порциями. Размер порции выбирается из следующих соображений:

1) время расчета одной "порции" должно быть много больше латентности MPI-системы,

2) общее число элементов в таблице, деленное на размер одной "порции", должно быть много больше числа процессов в MPI-задаче.

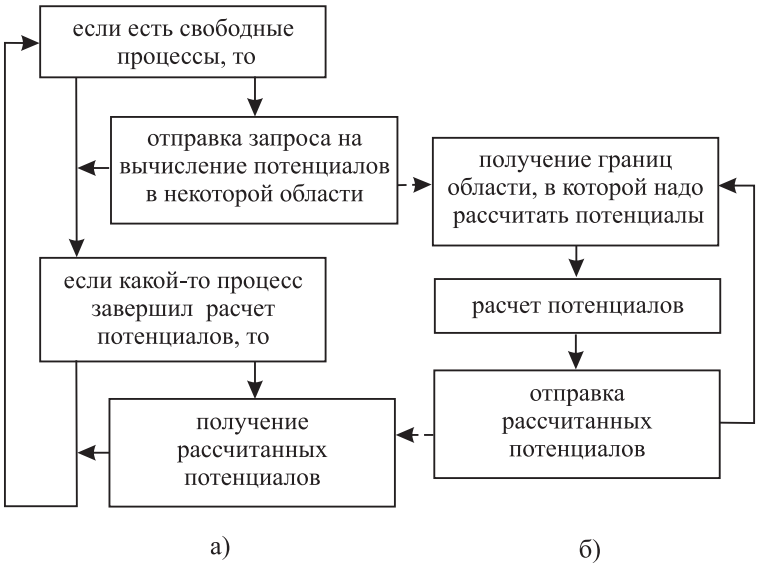

Рис. 3. Схема работы управляющего (а) и подчиненного (б) процессов программы SOLGRID

Первое требование позволяет снизить расходы на обмен данными между процессами, второе требование позволяет оптимально распределять расчеты по процессам вне зависимости от разброса скоростей их работы.

Область пространства, в которой требуется вычислить потенциалы, определяется первым и последним линейными индексами элементов в таблице. Линейный индекс il однозначно соответствует пространственным индексам ix, iy, iz и вычисляется по формуле $il = ix + 101 \times iy + 101 \times 101 \times iz$.

Схема работы подчиненного процесса показана на рис. 3б.

Так как управляющий процесс не занимается вычислениями потенциалов, то ускорение и эффективность работы параллельного SOLGRID рассчитывались по формулам $a(N) = \dfrac{t(2)}{t(N)}$, $e(N) = \dfrac{a(N)}{N-1}$.

Минимальное число ядер, которое можно выделить для работы SOLGRID, равно 2. На рис. 4 показаны зависимости ускорения и эффективности программы SOLGRID от числа процессов в диапазоне 1–128.

Как можно видеть из рис. 4, ускорение вычислений сетки потенциалов сохраняется почти линейным до 100 процессов. Таким образом, вычисление сетки потенциалов на 100 ядрах займет несколько минут вместо нескольких часов, требуемых для вычисления сетки потенциалов на одном ядре.

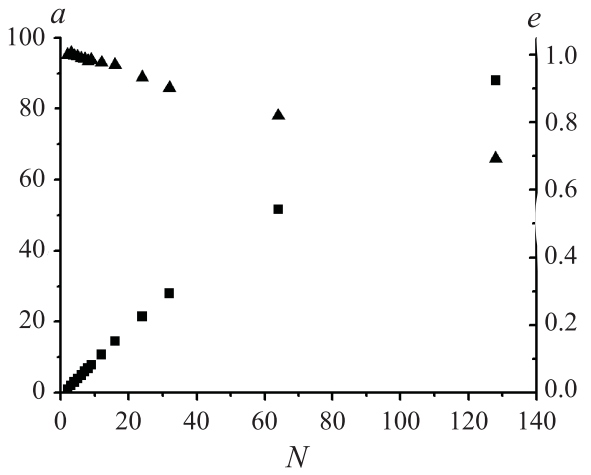

Рис. 4. Зависимости ускорения (a, квадратные точки) и эффективности (e, треугольные точки) вычислений от числа используемых ядер (N) для расчета сеток потенциалов

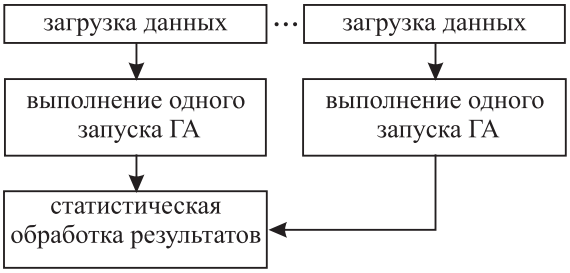

Рис. 5. Очевидная и простая схема организации параллельной работы программы SOL

5. Способы организации параллельного выполнения программы SOL. Как нетрудно заметить, независимые запуски ГА сами собой напрашиваются на параллельное выполнение. Каждый из этих запусков ГА длится примерно одно и то же время, никак не используя результаты других запусков ГА. Возможная схема работы параллельной программы показана на рис. 5.

Такая параллельная программа весьма проста и эффективно использует вычислительные ресурсы (то есть поток MPI-сообщений крайне невелик). Но у нее есть два недостатка. Первый из них заключается в том, что нельзя использовать вычислительных ядер больше, чем NOR (Number of Runs, число независимых запусков генетического алгоритма). Второй заключается в неделимости выполнения одного запуска ГА между вычислительными ядрами. То есть нельзя выполнить 50 независимых запусков ГА на 49 ядрах за 50/49 от времени выполнения одного запуска ГА. На каком-то из ядер придется выполнить два запуска ГА целиком, и суммарное время работы программы будет равно удвоенному времени выполнения одного запуска ГА. Для решения первой проблемы необходимо, чтобы обработка одного запуска ГА могла параллельно осуществляться несколькими ядрами, а для решения второй проблемы необходимо более гибко распределять вычисления независимых запусков ГА, чем это было описано выше. Иными словами, задача параллельного выполнения SOL складывается из двух иерархически связанных подзадач: подзадачи распределения вычислений независимых запусков ГА на верхнем уровне и подзадачи параллельного выполнения одного запуска ГА на нижнем уровне.

5.1. Параллельное выполнение одного запуска ГА. Около 80%–90% всего времени работы ГА занимают процедуры пересчета геномного представления положения лиганда в декартовы координаты и расчета энергии связывания с белком по получающимся координатам. На каждом шаге эволюции эти процедуры выполняются для большого набора особей, что позволяет использовать параллельные вычисления, в которых каждое из ядер рассчитывает энергию для своей части популяции. Аналогично можно параллельно выполнять процедуры набора особей в популяцию нового поколения. “Узкими” местами, где требуется синхронизация процессов, являются процедура нахождения лучшей особи по всей популяции и процедура отбора родителей следующего поколения.

Популяция может храниться распределенно по процессам, а может содержаться у одного процесса целиком. В данной работе были реализованы оба таких варианта хранения популяции.

5.1.1. Параллельное выполнение одного запуска ГА с распределенным хранением популяции. Каждый из процессов хранит и обрабатывает часть популяции. Схема этого алгоритма приведена на рис. 6.

В данной схеме организации параллельной работы ГА управляющие процессы слабо отличаются от подчиненных и выполняют "полезные" вычисления. Поэтому $a(N) = \dfrac{t(1)}{t(N)}$ и $e(N) = \dfrac{a(N)}{N}$. Зависимости ускорения и эффективности описанного алгоритма от числа процессов приведены на рис. 7.

Как можно заметить из рис. 7, эффективность алгоритма быстро спадает до числа ядер, равного 8, затем медленно спадает с 0.45 до 0.35.

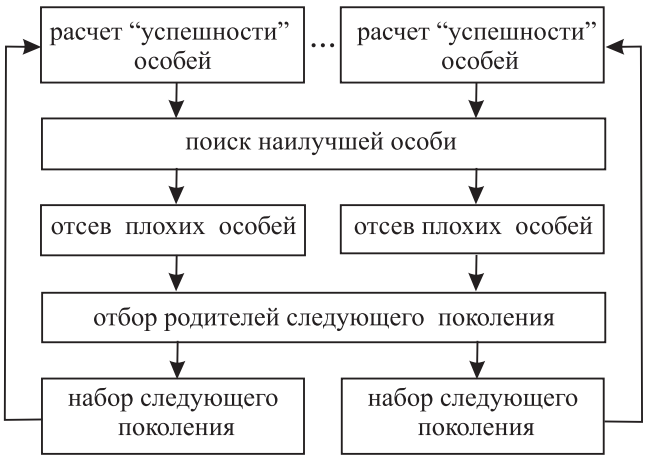

Рис. 6. Алгоритм параллельной работы ГА с распределенным хранением популяции. Показано два процесса. Один из процессов является управляющим. Отличия в его работе от работы подчиненных процессов незначительны

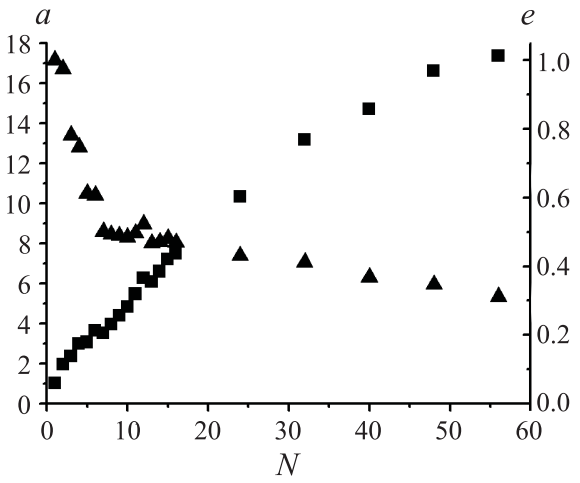

Рис. 7. Зависимости ускорения (a, квадратные точки) и эффективности (e, треугольные точки) вычислений от числа используемых ядер (N) для одного запуска ГА с распределенным хранением популяции на средней задаче

Заметно, что на участке N от 1 до 8 эффективность спадает как бы ступеньками: $N = 1$ и $N = 2$ — первая ступенька, $N = 3$ и $N = 4$ — вторая ступенька и так далее. Это связано с тем, что процессы на одном вычислительном узле конкурируют друг с другом за доступ к памяти. Дополнительные измерения показали, что вычислительный узел при полной загрузке всех восьми его ядер последовательными программами докинга считает в среднем в 1.9 раз медленнее, чем тот же узел при загрузке только одного ядра. Этот эффект взаимозамедления работы ядер также приводит к неодинаковой скорости работы процессов. Поэтому априорно равное распределение популяции между процессами становится неоптимальным.

Кроме того, следует добавить, что алгоритм не защищен от задействования чрезмерного количества ядер на вычисление одного запуска ГА. При неблагоприятном стечении обстоятельств может сложиться ситуация, когда время работы программы будет расти с увеличением числа задействованных ядер. Ситуация становится неблагоприятной тогда, когда лиганд простой, т.е. содержит мало атомов и мало торсионных степеней свободы, и латентность MPI-сети становится сравнимой со временем локальных расчетов. В данной работе в качестве "простого" лиганда было выбрано соединение с пятью торсионными степенями свободы и 23 атомами. Для сравнения, в качестве "среднего" лиганда использовалось соединение с семью торсионными степенями свободы и 48 атомами. Зависимость в подобных условиях ускорения от числа процессов приведена на рис. 10 (квадратные точки).

Как видно в данном случае, использование более 60 ядер не оправдано. Поэтому требуется ограничивать число используемых ядер на один запуск ГА, причем это ограничение следует выбирать с запасом, так как оно трудно предсказуемо для произвольно взятого лиганда. При превышении "рекомендованного" числа ядер производительность алгоритма резко падает.

В данном подходе все стадии алгоритма выполняются в той или иной степени параллельно, но выгода от параллельного отбора родителей следующего поколения довольно сомнительна. Она требуется скорее для того, чтобы сохранить концепцию распределенно хранимой популяции. Такой способ отбора родителей заставляет обмениваться сообщениями все процессы с управляющим процессом при отборе каждого нового родителя, что является главным фактором снижения производительности алгоритма. Другой фактор

снижения производительности — необходимость синхронизации процессов на каждом шаге эволюции, что заставляет весь алгоритм работать со скоростью самого медленного процесса.

5.1.2. Параллельное выполнение одного запуска ГА с локальным хранением популяции. Вместо того чтобы размещать популяцию в памяти нескольких процессов, что было проделано в предыдущем алгоритме, можно вообще отказаться от принципа распределенного хранения популяции, выделив один процесс, ответственный за всю популяцию. В пользу такого подхода можно добавить, что последовательно выполняемыми операциями остаются только отбрасывание плохих особей и отбор родителей нового поколения, что составляет весьма незначительный процент от времени всех вычислений. Кроме того, как можно было убедиться в предыдущем алгоритме, польза от параллельного отбора родителей следующего поколения сомнительна.

В основе идеи данного параллельного алгоритма лежит то, что вся популяция следующего поколения может быть воссоздана по небольшому массиву родителей следующего поколения. Тогда пусть подчиненные процессы создают популяцию по массиву родителей следующего поколения и время от времени отсылают результаты управляющему процессу. После накопления управляющим процессом популяции требуемого размера он отбирает новых родителей следующего поколения и рассылает их подчиненным процессам. Управляющий процесс также принимает участие в заполнении популяции наравне с подчиненными процессами. Схема такого алгоритма приведена на рис. 8.

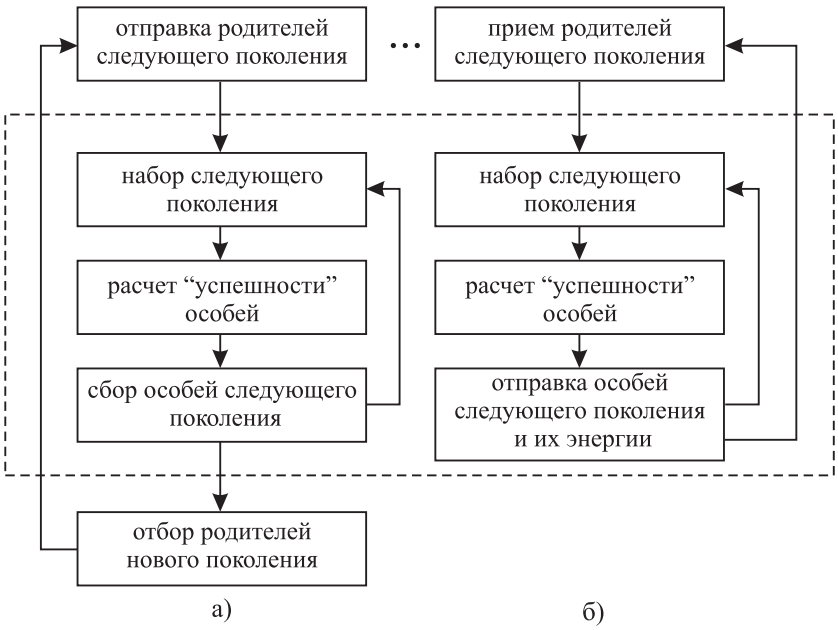

Рис. 8. Алгоритм параллельной работы ГА с локальным хранением популяции: а) схема управляющего процесса, б) схема одного из подчиненных процессов

Слева показана схема управляющего процесса, а справа — схема одного из подчиненных процессов. Операции, обведенные в пунктирный прямоугольник, выполняются до набора заданного числа особей в популяции. Вычисление и отправка особей выполняется небольшими порциями, много меньшими размера популяции.

Данный алгоритм "готов" к тому, что разные ядра могут работать с разной производительностью и, в отличие от предыдущего алгоритма, отбор родителей следующего поколения выполняется одним процессом целиком.

Значения ускорения получились немного меньше, чем у предыдущего алгоритма, зато нет такого быстрого снижения производительности при чрезмерном увеличении количества задействованных ядер в "неблагоприятных" условиях (см. сравнение на рис. 10). Это позволит использовать алгоритм на большем разнообразии кластерных систем.

В этой схеме организации параллельного выполнения одного запуска ГА отсутствуют управляющие процессы, т.е. ускорение и эффективность вычисляются по формулам: $a(N) = \dfrac{t(1)}{t(N)}$ и $e(N) = \dfrac{a(N)}{N}$. Зависимость ускорения и эффективности описанного алгоритма от числа процессов показана на рис. 9. Зависимость ускорения от числа процессов в неблагоприятном случае показана на рис. 10 круглыми точками.

Эффективность такого параллельного ГА определяется в первую очередь тем, что размер массива родителей следующего поколения много меньше, чем размер популяции.

Также такой подход позволяет естественным образом сосредоточить всю информацию о поколении в массиве родителей, что делает возможным обмен разными независимыми запусками ГА между группами процессов. Это становится важным при планировании вычислений сразу нескольких независимых запусков ГА.

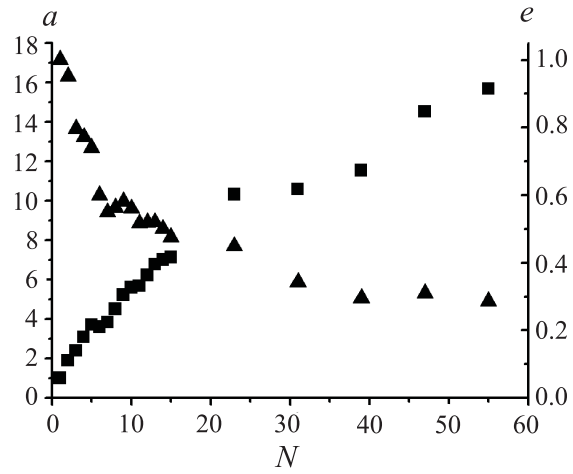

Рис. 9. Зависимости ускорения (a, квадратные точки) и эффективности (e, треугольные точки) вычислений от числа используемых ядер (N) для одного запуска ГА с локальным хранением популяции на средней задаче

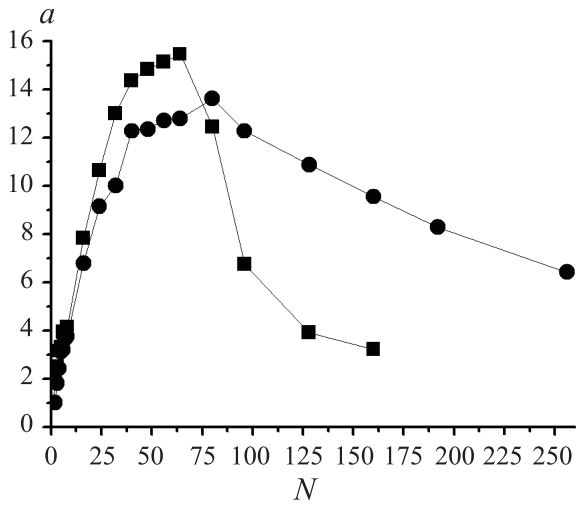

Рис. 10. Ускорения вычислений (a) в зависимости от числа используемых ядер (N) для одного запуска ГА с распределенным хранением популяции (квадратные точки) и для одного запуска ГА с локальным хранением популяции (круглые точки) в неблагоприятных условиях. Точки соединены линиями для удобства восприятия

5.2. Распределение вычислений независимых запусков ГА. В задаче докинга требуется выполнить несколько независимых запусков ГА. В связи с этим возникает вопрос: если, например, NOR равно 8 и число ядер равно 8, то лучше выполнить по одному независимому запуску ГА на каждом ядре или же каждый независимый запуск ГА, один за другим, посчитать параллельным образом на 8 ядрах? Очевидно, что первый вариант будет опережать второй по времени счета и простоте реализации. Ситуация становится более сложной, когда надо посчитать 50 независимых запусков ГА на 48 ядрах.

Если исходить из предположения, что один независимый запуск ГА неделимым образом выполняется группой процессов от начала до конца, то требуется спланировать разбиение всех процессов на группы и раздать им соответствующее число независимых запусков ГА так, чтобы последние независимые запуски ГА завершились одновременно.

Один подход заключается в том, чтобы разбить процессы на неравные группы, затем большим группам процессов дать на выполнение больше независимых запусков ГА, чем меньшим. Например, для выполнения 3 независимых запусков на 5 ядрах можно на двух ядрах выполнить 1 независимый запуск, а на трех других ядрах — выполнить 2 независимых запуска один за другим.

Другой подход заключается в том, чтобы запускать независимые запуски ГА "порциями" с переразбиением процессов на группы перед каждой "порцией". Например, для выполнения 6 независимых запусков на 8 ядрах можно сначала одновременно выполнить 4 независимых запуска на 4 группах по 2 процесса, затем перегруппировать процессы в две группы по 4 процесса, затем выполнить одновременно 2 независимых запуска на двух группах по 4 процесса.

Эти два подхода, очевидно, можно комбинировать. В программе SOL реализован второй подход, т.е. каждая группа процессов выполняет один независимый запуск ГА, затем процессы перегруппировываются. Этот подход можно применить для обоих описанных способов параллельной обработки одного запуска ГА.

Если же предполагать, что выполнение независимых запусков ГА можно переключать между разными группами процессов, то проблема некратности NOR и числа ядер во многом снимается. Главное — это равномерно выполнять все запуски ГА, перераспределяя их между группами процессов с тем расчетом, чтобы одновременно завершить их. Такой подход можно применить только для способа параллельной обработки одного запуска ГА с локальным хранением популяции.

Рассмотрим теперь эти два способа выполнения независимых запусков ГА: способ с неделимым выполнением одного запуска ГА и способ с переключаемым выполнением одного запуска ГА.

5.2.1. Выполнение независимых запусков ГА "порциями" по одному запуску ГА на группу

процессов. Схема такого алгоритма показана на рис. 11.

В этом алгоритме требуется как-то вычислять оптимальное значение K, где K — количество одновременно выполняемых независимых запусков ГА. Для этого необходимо задать некоторую оценочную функцию для $a(N)$ — ускорения параллельного выполнения одного запуска ГА на N ядрах. В программе принята такая модель этой функции: $a_{\text{est}}(N) = \dfrac{333N + 33(N-1)2}{236 + 97N}$ для $1 \leqslant N < 30$ и $a_{\text{est}}(N) = 0.5$ при $N \geqslant 30$.

Данная формула аппроксимирует экспериментально полученные значения ускорений для одного запуска ГА, а также в нее заложено ограничение, что на один запуск ГА надо выделять не более 29 процессов.

Эта формула $a_{\text{est}}(N)$ используется для оценки времени, требуемого на вычисление одного запуска ГА при помощи N процессов. Пусть $T_{\min}(NOR)$ — минимально возможное время в рамках описываемой стратегии, требуемое для вычисления NOR независимых запусков ГА на N ядрах. Нетрудно показать, что справедлива следующая формула:

$$T_{\min}(NOR) = \min_{\substack{K \geqslant 1,\\ K \leqslant NOR,\\ K \leqslant N}} \left\{ \frac{1}{a_{\text{est}}\big([N/K]\big)} + T_{\min}(NOR - K) \right\}.$$

Эта рекурсивная формула ограничена значением $T_{\min}(0) = 0$.

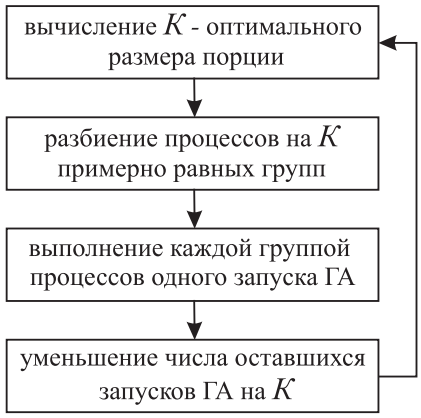

Рис. 11. Схема выполнения независимых запусков ГА “порциями” по одному запуску ГА на группу процессов

В таблице приведен пример планирования вычислений при помощи данного подхода. Требуемое число независимых запусков ГА равно 50. Во второй колонке таблицы показаны “порции” одновременно отправляемых на выполнение запусков ГА. Все независимые запуски ГА из одной “порции” вычисляются параллельно, и следующая “порция” запускается только после полного выполнения предыдущей.

Разбиение имеющихся 50 запусков ГА на порции одновременно выполняемых запусков ГА

Число ядер	Количество запусков ГА в “порциях”	Время расчетов в условных единицах
30	$15 + \ \ 5 + 30$	$1/a_{\text{est}}(2) + 1/a_{\text{est}}(6) + 1/a_{\text{est}}(1)$
49	$2 + 48$	$1/a_{\text{est}}(24) + 1/a_{\text{est}}(1)$
70	$15 + 35$	$1/a_{\text{est}}(4) + 1/a_{\text{est}}(2)$
256	50	$1/a_{\text{est}}(5)$

Такой способ выполнения нескольких независимых запусков ГА был объединен с первым способом параллельного выполнения одного запуска ГА, в котором популяция хранится распределенно. Ускорение и эффективность такого “двухуровневого” параллельного алгоритма представлена на рис. 12. В этой схеме параллельной работы SOL отсутствуют управляющие процессы (т.е. все функции по управлению занимают пренебрежимо малое время и выполняются в вычислительных процессах в промежутках между запусками ГА). Поэтому ускорение и эффективность вычисляются по формулам $a(N) = t(1)/t(N)$ и $e(N) = a(N)/N$.

Может показаться странным, что 50 ядер не могут выполнить 50 независимых запусков ГА в 50 раз быстрее одного ядра. В данном случае необходимо вспомнить об эффекте конкурирующего доступа к оперативной памяти у ядер одного вычислительного узла.

Хорошо заметна немонотонность роста $a(N)$ при больших N. Это связано с неделимостью выполнения одного запуска ГА, что приводит к простаиванию некоторых ядер из-за некратности числа ядер и числа независимых запусков ГА.

Еще одним недостатком данного метода параллельной работы программы SOL является необходимость иметь предсказываемые значения ускорений одного запуска ГА, заданные в программе или в ее параметрах. Эти значения будут разными для разных кластерных систем.

5.2.2. Выполнение независимых запусков ГА с переключениями между группами процессов. Второй способ параллельного выполнения одного запуска ГА, с локальным хранением популяции,

имеет важную особенность: выполнение одного запуска ГА может быть приостановлено, а вместо него начато выполнение другого запуска ГА. Подобная возможность приводит к следующей идее выполнения многих независимых запусков ГА. Пусть один процесс хранит массивы родителей следующего поколения для всех запусков ГА. Назовем этот процесс нулевым. Остальные процессы разбиты на группы, каждая группа обрабатывает один запуск ГА. Каждая группа процессов получает от нулевого массив родителей следующего поколения, выполняет одно или несколько поколений, затем возвращает новый массив родителей нулевому процессу. Таким образом, нулевой процесс может обменивать выполнение независимых запусков ГА между разными группами процессов. Каждая из групп процессов может работать асинхронно с прочими группами процессов. Нулевой процесс не участвует в "полезных" расчетах. Для эффективного обмена независимыми запусками ГА между группами процессов необходимо, чтобы количество групп процессов было меньше, чем количество независимых запусков ГА. В программе SOL задано, чтобы количество групп процессов составляло 80% от количества независимых запусков ГА. Группы процессов совсем не обязательно должны быть равными, но являются почти равными для максимизации общей эффективности. Схема управления независимыми запусками показана на рис. 13.

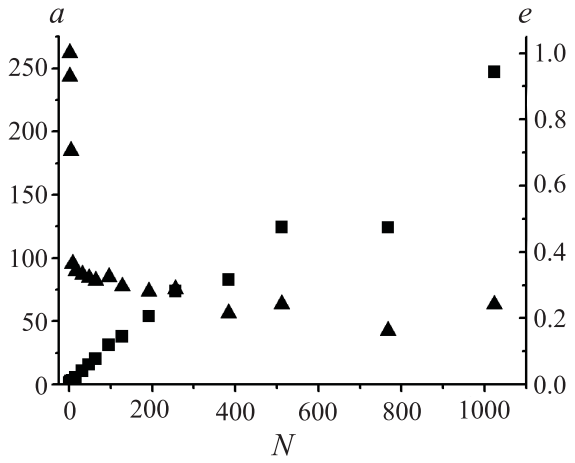

Рис. 12. Зависимости ускорения (a, квадратные точки) и эффективности (e, треугольные точки) вычислений от числа используемых ядер (N) для 50 запусков ГА с распределенным хранением популяции на средней задаче

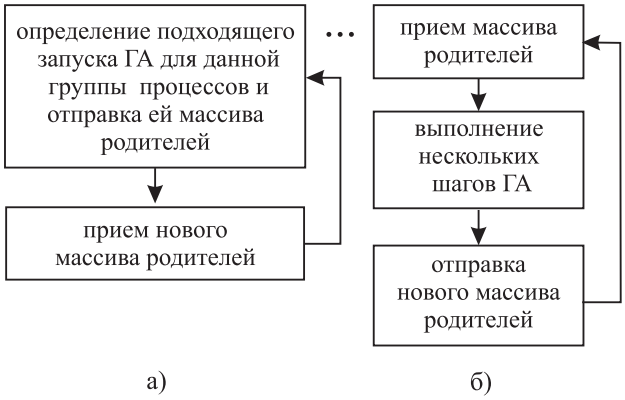

Рис. 13. Схема выполнения независимых запусков ГА с переключениями между группами процессов: а) алгоритм нулевого процесса, б) алгоритм одной из групп процессов

Нулевой процесс так распределяет независимые запуски между группами процессов, чтобы все независимые запуски выполнялись с одинаковой средней скоростью. Сделать это можно, запоминая, сколько каждая группа процессов суммарно выполнила шагов ГА, таким образом, вычисляя удельную производительность каждой из групп процессов. Кроме того, по степени завершенности запусков ГА можно определить удельную долю от всех оставшихся вычислений в этом запуске ГА. Тогда для группы процессов с удельной скоростью работы v подбирается запуск ГА с удельной долей оставшихся вычислений, максимально близкой к v. Для возможности такого выбора и было оговорено, что 20% от всех запусков ГА будут находиться в резерве у нулевого процесса.

Данный способ параллельной работы SOL показал следующие характеристики. Так как один процесс (нулевой) не выполняет вычислений в рамках ГА, то при вычислении ускорения и эффективности этот процесс не рассматривался: $a(N) = \dfrac{t(2)}{t(N)}$ вместо $a(N) = \dfrac{t(1)}{t(N)}$ и $e(N) = \dfrac{a(N)}{N-1}$ вместо $e(N) = \dfrac{a(N)}{N}$. Зависимости ускорения и эффективности от числа процессов показаны на рис. 14.

Мы видим, что зависимость $a(N)$ более монотонна, чем для предыдущей стратегии управления независимыми запусками ГА. Для данного алгоритма конкретные характеристики кластера не столь существенны, распределение нагрузки между группами процессов, а также нагрузки внутри группы процессов, автоматически выравнивается пропорционально производительности.

Однако в данном алгоритме налагается больше условий на параллельное выполнение одного запуска ГА. Это выполнение должно быть таковым, чтобы его можно было легко приостанавливать и переключать на выполнение другого запуска ГА. Кроме того, потребовался один нулевой процесс, не выполняющий

непосредственно вычислений в рамках ГА, а занимающийся непрерывно рассылкой и приемом массивов родителей следующего поколения.

6. Выводы. Рассмотрены возможности параллельной реализации программ докинга SOLGRID и SOL. Обычно при докинге требуется построить сетку для одного белка, и она будет использоваться для докинга многих лигандов (так называемый "виртуальный скрининг"). Наличие параллельной программы SOLGRID позволяет существенно уменьшить время построения сетки потенциалов от нескольких часов до нескольких минут, поскольку полученное ускорение вычисления сетки потенциалов сохраняется почти линейным до 100 и более процессов. Заметим, что сетка потенциалов будет рассчитана непосредственно на кластере, поэтому можно избежать переписывания сетки из внешнего источника на кластер по, как правило, медленному каналу связи. Таким образом, задержка с момента получения доступа к кластеру до начала непосредственной работы программы SOL, связанная с получением сетки потенциалов на кластере, может быть сведена к нескольким минутам.

Эффективность двух рассмотренных в данной работе параллельных вариантов программы SOL вначале падает, когда число задействованных ядер меняется от 1 до 64, т.е. линейного скалирования в этом диапазоне не наблюдается. Затем, когда число задействованных ядер меняется от 128 до 1024, появляется линейная скалируемость обоих параллельных вариантов программы SOL. Возможно, что линейность скалирования будет сохраняться и далее при росте числа задействованных ядер. Для первого параллельного варианта программы SOL с распределенным хранением популяции и с непереносимым выполнением одного запуска ГА между группами процессов эффективность использования вычислительных ресурсов в этих условиях составляет 20%–30%. Для второго параллельного варианта программы SOL с локальным хранением популяции и переносимым выполнением одного запуска ГА между группами процессов эффективность использования вычислительных ресурсов в этих условиях составляет 30%–40%. Эффективность работы второго параллельного варианта программы SOL выше, чем у первого, за счет более гибкого алгоритма параллельного выполнения нескольких независимых запусков ГА, хотя один независимый запуск ГА немного эффективнее выполняется в первом варианте программы SOL. В рабочих режимах работы программы SOL (до 32 ядер на один независимый запуск ГА) конкретный вид лиганда, в том числе количество атомов и торсионных степеней свободы в его составе, слабо влияют на ускорение и эффективность работы параллельных версий SOL.

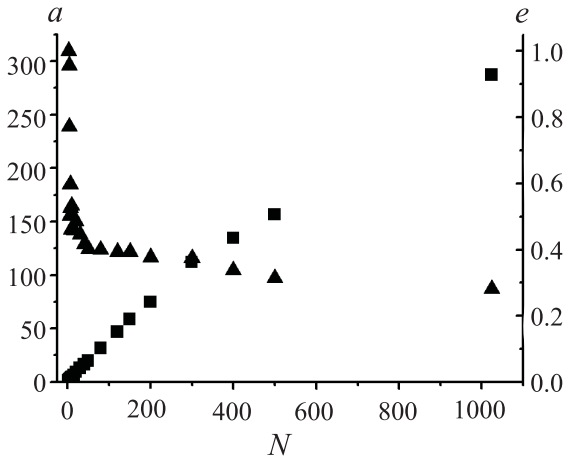

Рис. 14. Зависимости ускорения (a, квадратные точки) и эффективности (e, треугольные точки) вычислений от числа используемых ядер (N) для 50 запусков ГА с локальным хранением популяции на средней задаче

Полезность использования параллельной программы SOL при виртуальном скрининге сомнительна, так как обычно число имеющихся лигандов (от нескольких тысяч до миллионов) превосходит число доступных для данного пользователя процессоров в кластере. Поэтому в таких условиях эффективнее запускать множество последовательных версий SOL, каждая из которых будет заниматься докингом одного лиганда.

Применение параллельной программы SOL оправдано в тех случаях, когда требуется провести докинг одного лиганда или небольшого их числа. В этом случае время ожидания результатов может быть уменьшено в несколько десятков или сотен раз, пусть и за счет несколько неоптимального использования ресурсов кластерного суперкомпьютера.

В данной работе не затрагивалась возможность изменения самого алгоритма докинга в параллельной версии программы SOL, чтобы была возможность сравнения производительности алгоритмов, обеспечивающих одинаковое качество докинга. Поэтому в параллельной версии программы SOL осталась возможность улучшения эффективности работы за счет изменения самого алгоритма докинга.

СПИСОК ЛИТЕРАТУРЫ

1. *Садовничий В.А., Сулимов В.Б.* Суперкомпьютерные технологии в медицине // Суперкомпьютерные технологии в науке, образовании и промышленности / Под ред. В.А. Садовничего, Г.И. Савина, Вл.В. Воеводина. М: Изд-во Моск. ун-та, 2009. 16–23.
2. *Zoete V., Grosdidier A., Michielin O.* Docking, virtual high throughput screening and in silico fragment-based drug design // J. Cell. Mol. Med. 2009. **13**. 238–248.

3. *Klebe G.* Virtual ligand screening: strategies, perspectives and limitations // Drug Disc. Today. 2006. **11**. 580–594.
4. *Wlodawer A., Vondrasek J.* Inhibitors of HIV-1 protease: a major success of structure-assisted drug design // Annu. Rev. Biophys. Biomol. Struct. 1998. **27**. 249–284.
5. *Tarle I., Borhani D.W., Wilson D.K., Quiocho F.A., Petrash J.M.* Probing the active site of human aldose reductase. Site-directed mutagenesis of Asp-43, Tyr-48, Lys-77, and His-110 // J. Biol. Chem. 1993. **268**, N 34. 25687–25693.
6. *Lin J.H., Perryman A.L., Schames J.R., McCammon J.A.* Computational drug design accommodating receptor flexibility: the relaxed complex scheme // J. Am. Chem. Soc. 2002. **124**, N 20. 5632–5633.
7. *Erickson J.A., Jalaie M., Robertson D.H., Lewis R.A., Vieth M.* Lessons in molecular recognition: the effects of ligand and protein flexibility on molecular docking accuracy // J. Med. Chem. 2004. **47**, N 1. 45–55.
8. *Kuntz I.D., Blaney J.M., Oatley S.J., Langridge R., Ferrin T.E.* A geometric approach to macromolecule-ligand interactions // J. Mol. Biol. 1982. **161**. 269–288.
9. *Chen R., Weng Z.* A novel shape complementarity scoring function for protein-protein docking // Proteins. 2003. **51**, N 3. 397–408.
10. *Ritchie D.W., Kozakov D., Vajda S.* Accelerating and focusing protein-protein docking correlations using multi-dimensional rotational FFT generating functions // Bioinformatics. 2008. **24**, N 17. 1865-1873.
11. *Morris G.M., Goodsell D.S., Halliday R.S., Huey R., Hart W.E., Belew R.K., Olson A.J.* Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function // J. Comput. Chem. 1998. **19**. 1639–1662.
12. *Huey R., Morris G.M., Olson A.J., Goodsell D.S.* A semiempirical free energy force field with charge-based desolvation // J. Comput. Chem. 2007. **28**. 1145–1152.
13. *Ewing T.J., Makino S., Skillman A.G., Kuntz I.D.* DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases // J. Comput. Aided Mol. Des. 2001. **15**. 411–428.
14. *Abagyan R., Totrov M., Kuznetsov D.* ICM — a new method for protein modeling and design: applications to docking and structure prediction from the distorted native conformation // J. Comput. Chem. 1994. **15**. 488–506.
15. *Jones G., Willett P., Glen R.C., Leach A.R., Taylor R.* Development and validation of a genetic algorithm for flexible docking // J. Mol. Biol. 1997. **267**. 727–748.
16. *Verdonk M.L., Cole J.C., Hartshorn M.J., Murray C.W., Taylor R.D.* Improved protein-ligand docking using GOLD // Proteins. 2003. **52**. 609–623.
17. *Романов А.Н., Кондакова О.А., Григорьев Ф.В., Сулимов А.В., Лущекина С.В., Мартынов Я.Б., Сулимов В.Б.* Компьютерный дизайн лекарственных средств: программа докинга SOL // Вычислительные методы и программирование. 2008. **9**, № 2. 64–84.
18. *Cavasotto C.N., Abagyan R.A.* Protein flexibility in ligand docking and virtual screening to protein kinases // J. Mol. Biol. 2004. **337**. 209–225.
19. *Rarey M., Kramer B., Lengauer T., Klebe G.* A fast flexible docking method using an incremental construction algorithm // J. Mol. Biol. 1996. **261**. 470–489.
20. *Claussen H., Buning C., Rarey M., Lengauer T.* FlexE: efficient molecular docking considering protein structure variations // J. Mol. Biol. 2001. **308**. 377–395.
21. *Sousa S.F., Fernandes P.A., Ramos M.J.* Protein-ligand docking: current status and future challenges // Proteins. 2006. **65**. 15–26.
22. *Halperin I., Ma B., Wolfson H., Nussinov R.* Principles of docking: an overview of search algorithms and a guide to scoring functions // Proteins. 2002. **47**, N 4. 409–443.
23. *Aqeel A., Sina K., Holger G.* Protein flexibility and mobility in structure-based drug design // Frontiers in Drug Design & Discovery. 2007. **3**. 455–476.
24. *Sheng-You Huang, Xiaoqin Zou.* Advances and challenges in protein-ligand docking // Int. J. Mol. Sci. 2010. **11**. 3016–3034.
25. *Miller M.D., Kearsley S.K., Underwood D.J., Sheridan R.P.* FLOG: a system to select "quasi-flexible" ligands complementary to a receptor of known three-dimensional structure // J. Comput. Aided Mol. Des. 1994. **8**, N 2. 153–174.
26. *Friesner R.A., Banks J.L., Murphy R.B., Halgren T.A., Klicic J.J., Mainz D.T., Repasky M.P., Knoll E.H., Shelley M., Perry J.K., Shaw D.E., Francis P., Shenkin P.S.* Glide: a new approach for rapid, accurate docking and scoring. 1. Method and assessment of docking accuracy // J. Med. Chem. 2004. **47**. 1739–1749.
27. *Halgren T.A., Murphy R.B., Friesner R.A., Beard H.S., Frye L.L., Pollard W.T., Banks J.L.* Glide: a new approach for rapid, accurate docking and scoring. 2. Enrichment factors in database screening // J. Med. Chem. 2004. **47**. 1750–1759.
28. *Friesner R.A., Murphy R.B., Repasky M.P., Frye L.L., Greenwood J.R., Halgren T.A.* Extra precision glide: docking and scoring incorporating a model of hydrophobic enclosure for protein-ligand complexes // J. Med. Chem. 2006. **49**. 6177–6196.
29. *Goldberg D.E.* Genetic algorithms in search, optimization, and machine learning. Reading: Addison-Wesley, 1989.
30. *Goldberg D.E.* Real-coded genetic algorithm, virtual alphabets, and blocking // Complex systems. 1991. **5**. 139–167.
31. *Konfrst Z.* Parallel genetic algorithms: advances, computing trends, applications and Perspectives // Proc. 18th Int. Parallel and Distributed Processing Symposium (IPDPS'04). Workshop 6. 26–30 April 2004. New Mexico. USA. **7**,

162. Santa Fe, 2004.
32. *Thomsen R., Christensen M.H.* MolDock: a new technique for high-accuracy molecular docking // J. Med. Chem. 2006. **49**. 3315–3321.
33. *McMartin C., Bohacek R.S.* QXP: powerful, rapid computer algorithms for structurebased drug design // J. Comput. Aided Mol. Des. 1997. **11**. 333–344.
34. *Namasivayam V., Gunther R.* pso@autodock: a fast flexible molecular docking program based on swarm intelligence // Chem. Biol. Drug Des. 2007. **70**, N 6. 475–484.
35. *Chen H.M., Liu B.F., Huang H.L., Hwang S.F., Ho S.Y.* SODOCK: swarm optimization for highly flexible protein-ligand docking // J. Comput. Chem. 2007. **28**, N 2. 612–623.
36. *Kirkpatrick S., Gelatt C.D., Vecchi M.P.* Optimization by simulated annealing // Science. 1983. **220**, N 4598. 671–680.
37. *Dias R., de Azevedo W.F.* Molecular docking algorithms // Curr. Drug Targets. 2008. **9**, N 12. 1040-1047.
38. *Bursulaya B.D., Totrov M., Abagyan R., Brooks C.L.* Comparative study of several algorithms for flexible ligand docking // J. Comput. Aided Mol. Des. 2003. **17**, N 11. 755–763.
39. *Wang R., Wang S.* How does consensus scoring work for virtual library screening? An idealized computer experiment // J. Chem. Inf. Comput. Sci. 2001. **41**, N 5. 1422–1426.
40. *Teramoto R., Fukunishi H.* Supervised consensus scoring for docking and virtual screening // J. Chem. Inf. Model. 2007. **47**, N 2. 526–534.
41. *Congreve M., Chessari G., Tisi D., Woodhead A.J.* Recent developments in fragment-based drug discovery // J. Med. Chem. 2008. **51**, N 13. 3661–3680.
42. *Pegg S.C., Haresco J.J., Kuntz I.D.* A genetic algorithm for structure-based de novo design // J. Comput. Aided Mol. Des. 2001. **15**, N 10. 911–933.
43. *Moustakas D.T., Lang P.T., Pegg S., Pettersen E., Kuntz I.D., Brooijmans N., Rizzo R.C.* Development and validation of a modular, extensible docking program: DOCK 5 // J. Comput. Aided Mol. Des. 2006. **20**, N 10-11. 601–619.
44. *Levesque M.J., Ichikawa K., Date S., Haga J.H.* Design of a grid service-based platform for in silico protein-ligand screenings // Comput. Methods Programs Biomed. 2009. **93**, N 1. 73–82.
45. *Morris G.M., Huey R., Lindstrom W., Sanner M.F., Belew R.K., Goodsell D.S., Olson A.J.* AutoDock4 and AutoDockTools4: automated docking with selective receptor flexibility // J. Comput. Chem. 2009. **30**. 2785–2791.
46. FightAIDS@Home (http://fightaidsathome.scripps.edu).
47. Discover Dengue Drugs–Together (http://www.worldcommunitygrid.org/research/dddt/overview.do).
48. *Zhang Sh., Kumar K., Jiang X., Wallqvist A., Reifman J.* DOVIS: an implementation for high-throughput virtual screening using AutoDock // BMC Bionformatics 8. 2008. 126–129.
49. *Zhang Sh., Kumar K., Hu X., Wallqvist A., Reifman J.* DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0 // Chem. Central J. 2008. **2**. 18–24.
50. *Vistoli G., Pedretti A., Mazzolari A., Testa B.* Homology modeling and metabolism prediction of human carboxylesterase-2 using docking analyses by GriDock: a parallelized tool based on AutoDock 4.0 // J. Comput. Aided Mol. Des. 2010. **24**, N 9. 771–787.
51. *Morris G.M., Goodsell D.S., Huey R., Olson A.J.* Distributed automated docking of flexible ligands to proteins: parallel applications of AutoDock 2.4 // J. Comput. Aided Mol. Des. 1996. **10**, N 4. 293–304.
52. *Prakhov N.D., Chernorudskiy A.L., Gainullin M.R.* VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters // Bioinformatics. 2010. **26**. 1374–1375.
53. *Trott O., Olson A.J.* AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading // J. Comput. Chem. 2010. **31**, N 2. 455–461.
54. *Chang M.W., Ayeni C., Breuer S., Torbett B.E.* Virtual screening for HIV protease inhibitors: a comparison of AutoDock 4 and Vina // PLoS One. 2010. **5**, N 8. e11955.
55. *Khodade P., Prabhu R., Chandra N., Rahab S., Govindarajanb R.* Parallel implementation of AutoDock // J. Appl. Cryst. 2007. **40**. 598–599.
56. *Micevski D.* Optimizing Autodock with CUDA. Victorian Partnership For Advanced Computing Ltd, 2009.
57. http://gpuautodock.sourceforge.net/
58. *Halgren T.A.* Merck Molecular Force Field. I. Basis, form, scope, parametrization and performance of MMFF94 // J. Comput. Chem. 1996. **5&6**. 490–519; 520–552; 553–586; 587–615; 616–641.
59. *Ghosh A., Rapp C.S., Friesner R.A.* Generalized Born Model Based on a Surface Integral Formulation // J. Phys. Chem. B. 1998. **102**. 10983–10990.
60. *Romanov A.N., Jabin S.N., Martynov Y.B., Sulimov A.V., Grigoriev F.V., Sulimov V.B.* Surface Generalized Born method: a simple, fast and precise implicit solvent model beyond the Coulomb approximation // J. Phys. Chem. A. 2004. **108**. 9323–9327.

Поступила в редакцию
14.12.2010