Мультплатформенный программный комплекс для решения задач оптимизации в распределенной вычислительной среде *

М. А. Посыпкин

Центр Грид-технологий и распределенных вычислений Института системного анализа РАН

Проведен анализ существующих методов решения задач комбинаторных и оптимизационных задач. Рассмотрены точные, эвристические и комбинированные подходы. Исследованы различные архитектуры современных ЭВМ, многопроцессорные комплексы с общей и распределенной памятью. На основании проведенного анализа разработаны эффективные реализации методов решения задач глобальной оптимизации в средах параллельных и распределенных вычислений. Основным результатом работы являются методы и основанная на них иерархическая программная инфраструктура для решения комбинаторных и оптимизационных задач большой размерности в среде распределенных и параллельных вычислений. Эта инфраструктура позволяет решать задачи оптимизации на последовательных, параллельных системах с общей и распределенной памятью и грид-системах.

Введение

Задачи глобальной оптимизации применяются в различных областях современной науки и техники. Эти задачи являются чрезвычайно важными для развития экономики в целом и таких отраслей, как нанотехнологии, микроэлектроника, биология, легкая и тяжелая промышленность. Поэтому разработка эффективных методов решения таких задач и их реализация в виде прикладных программных комплексов представляется актуальной научной и практической проблемой. Отличительной особенностью задач

* Работа выполнена при поддержке Аналитической ведомственной программы «Развитие научного потенциала высшей школы» и Совета по грантам Президента Российской Федерации (№ НШ–5511.2008.9).

поиска глобального экстремума является высокая вычислительная сложность. Поэтому для решения таких задач зачастую применяются методы параллельных и распределенных вычислений.

В настоящее время наблюдается стремительное развитие распределенных и параллельных систем. Относительно недавно появились многоядерные процессоры, превратившие персональные компьютеры в настольные параллельные системы с общей памятью. Также быстро развивается сетевая архитектура, позволяющая выполнять вычисления в распределенной среде, состоящей из тысяч компьютеров, находящихся в географически удаленных точках. В последнее время широкое распространение получили различные типы Грид-систем, объединяющие суперкомпьютерные ресурсы или ресурсы рабочих станций различной ведомственной принадлежности и расположенные на большом расстоянии друг от друга.

Многообразие архитектур и базового программного обеспечения создает существенные трудности для программистов, которые вынуждены осваивать широкий спектр средств и методов разработки программ. Это приводит к потерям эффективности труда исследователей и разработчиков ПО, обусловленным необходимостью изучать непрофильные области знаний. В сложившейся ситуации становится очевидной потребность в программных средствах, облегчающих разработку приложений для решения задач оптимизации на различных типах распределенных систем.

В данной работе представлена иерархическая программная инфраструктура для решения задач оптимизации в среде параллельных и распределенных вычислений. Суть подхода состоит в реализации каркасных модулей, реализующих типовые вычислительные схемы оптимизационных алгоритмов на различных платформах. После этого, программа для решения конкретной задачи на заданной платформе получается объединением проблемно-зависимых модулей для этой задачи и проблемно-независимых модулей, реализующих общую схему для выбранной платформы. Такой подход позволяет сэкономить усилия разработчиков за счет повторного использования общих частей, имеющихся у методов решения различных задач оптимизации.

1. Задачи конечномерной оптимизации и методы их решения

Задача поиска глобального минимума (максимума) функции $f(x)$ на *допустимом множестве* $X \subseteq R^n$ состоит в отыскании такой точки $x_* \in X$, что $f(x_*) \le f(x)$ ($f(x_*) \ge f(x)$) для всех $x \in X$. Далее будем предполагать, что решается задача минимизации функции *f*. Ограничения, связанные с вычислительной погрешностью или недостатком ресурсов,

часто не позволяют найти точное решение данной задачи. В этом случае переходят к поиску приближенного решения, т. е. точки из множества ε-оптимальных решений $X_*^\varepsilon = \{x \in X : f(x) \le f(x_*) + \varepsilon\}$. Поиск точного решения можно рассматривать как частный случай поиска приближенного решения с $\varepsilon = 0$.

Способ задания множества X позволяет провести грубую классификацию задач. Если $X = R^n$, то задача относится к классу *задач безусловной оптимизации*. Если допустимое множество задано с помощью ограничений, то говорят об *условной оптимизации*. При этом, если X конечно, то рассматриваемая задача относится к области *дискретной оптимизации.*

Можно выделить два основных семейства методов решения задач конечномерной оптимизации: точные и эвристические. Точные методы позволяют гарантировать оптимальность найденного решения. К этому классу можно отнести различные варианты метода ветвей и границ, отсечений и др. Для точных методов характерна высокая трудоемкость, которая часто не позволяет применять их при решении реальных задач. Эвристические методы основаны на предположениях о свойствах оптимального решения. В отличие от точных, эвристические методы не дают гарантии оптимальности найденного решения. Однако в условиях ограниченности вычислительных ресурсов эвристики зачастую являются единственным способом нахождение решения. Также распространены гибридные методы, при которых эвристические методы применяются для нахождения решения, а точные — для доказательства оптимальности. Эффективность гибридных методов обусловлена тем, что эвристические алгоритмы нередко обладают более высокой скоростью сходимости к оптимуму по сравнению с точными методами.

Одной из наиболее распространенных схем организации точных методов является т. н. *метод ветвей и границ* (МВГ), основанный на разбиении допустимого множества. Приведем общую схему метода. На протяжении всего времени работы поддерживается список $\{X_i\}$ подмножеств допустимого множества. Первоначально он состоит из одного элемента — допустимого множества X. Далее выбирается один из элементов списка — подмножество X_i. Если X_i удовлетворяет *правилам отсева*, то выбранный элемент удаляется из списка. В противном случае он подвергается дроблению на более мелкие подмножества с помощью *правила декомпозиции*. Полученные подмножества замещают в списке выбранный элемент. Алгоритм завершает свою работу, когда в последовательности $\{X_i\}$ не остается ни одного элемента.

Правилом отсева будем называть любую функцию $\xi : \mathbf{S} \to \{0,1\}$, определенную на некотором множестве $\mathbf{S}$ подмножеств R^n, и сопоставляю-

щую каждому подмножеству из $\mathbf{S}$ число 0 либо 1. Если для некоторого $V \in \mathbf{S}$ выполняется $\xi(V) = 1$, то будем говорить, что подмножество V *удовлетворяет правилу отсева* ξ. В противном случае будем говорить, что V *не удовлетворяет правилу отсева* ξ.

Пусть заданы правила отсева $\Xi = \{\xi_1, \ldots, \xi_m\}$. Конечную совокупность подмножеств $\mathbf{C} = \{X_1, \cdots, X_t\}$, такую что $X \subseteq \bigcup_{i=1}^{t} X_i$ и для любого $j \in \{1, \ldots, t\}$ найдется хотя бы одно $i \in \{1, \ldots, m\}$, такое что $\xi_i(X_j) = 1$, будем называть *покрытием множества X* для набора правил отсева Ξ.

Приведем формулировку одного из наиболее часто применяемых правил отсева. Для этого понадобятся понятия оценки и рекорда. *Оценкой* называется функция $g : \mathbf{S} \to R$, определенная на множестве $\mathbf{S}$ подмножеств R^n, такая что для любого $V \in \mathbf{S}$ выполняется $g(V) \le \min_{x \in V \cap X} f(x)$. В качестве оценки, как правило, выбираются легко вычисляемые функции. *Рекордом* называют наименьшее значение целевой функции, найденное в процессе решения. Если $x_1, \ldots, x_k$ — последовательность точек, в которых вычислялось значение целевой функции, то $f_k = f(x_r) = \min_{i=1,\ldots,k} f(x_i)$ — рекорд, а x_r — рекордное решение. Правило отсева ξ_{rec} *по рекорду* формулируется следующим образом:

$$\xi_{rec}(V) = \begin{cases} 1, g(V) \ge f_k - \varepsilon, \\ 0, g(V) < f_k - \varepsilon. \end{cases}$$

Совокупность правил отсева должна выбираться таким образом, чтобы из существования покрытия следовало, что найденный рекорд является ε-оптимальным решением. Такую совокупность будем называть *корректной*. Например, совокупность $\{\xi_{rec}\}$, состоящая из одного правила отсева *по рекорду*, будет корректной.

В процессе нахождения рекорда последовательность точек $x_1, \ldots, x_k$, в которых вычисляется значения целевой функции, формируется на основе разбиваемых множеств. Например, если множества $\{X_i\}$ являются параллелепипедами, то в качестве элементов последовательности $x_1, \ldots, x_k$ берутся их центры. Также возможен подход, при котором последовательность точек $x_1, \ldots, x_k$ формируется независимым от построения покрытия способом с помощью различных эвристических процедур. В качестве эвристик могут применяться различные локальные алгоритмы (метод гради-

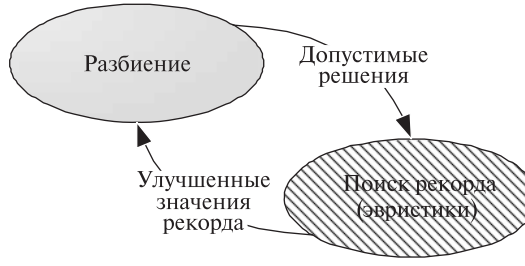

Рис. 1. Взаимодействие МВГ и эвристических процедур

ентного спуска, Ньютона, перебор в некоторой окрестности и т. п.), а также более сложные алгоритмы, например стохастические или генетические подходы. Таким образом, в решении задачи участвуют как алгоритмы, выполняющие разбиение, так и эвристические алгоритмы. Они взаимодействуют в соответствии со следующей схемой (рис. 1): в процессе разбиения допустимого множества генерируются точки, предоставляемые для обработки эвристическому алгоритму. Эвристический алгоритм обрабатывает предоставленные решения, полученный в результате улучшенный рекорд используется для усиления отсева при выполнении разбиений.

Последней составляющей метода является правило декомпозиции $\Delta : \mathbf{S} \to 2^{\mathbf{S}}$, которое является функцией, определенной на множестве $\mathbf{S}$ подмножеств R^n, и сопоставляющей некоторому подмножеству $V \in \mathbf{S}$ конечную совокупность его подмножеств $V_1,...,V_m$, при этом

$$V = \bigcup_{i=1}^{m} V_i,\ V_i \cap V_j = \varnothing \text{ для всех } 1 \le i < j \le m.$$

Совокупность правил отсева, правила декомпозиции и способ вычисления рекорда должны выбираться так, чтобы обеспечивать конечность числа шагов МВГ.

2. Основные типы архитектур современных вычислительных систем

2.1. Последовательные архитектуры

Классическая последовательная архитектура, предполагающая один поток команд и данных, в настоящее время не встречается даже на персональных компьютерах. Современные микропроцессоры, как правило, имеют несколько вычислительных ядер. При этом, каждое ядро можно рассматривать

как вычислитель с последовательной архитектурой. При планировании процессов и потоков операционная система рассматривает вычислительные ядра как независимые процессоры, не делая принципиальных различий между многоядерными и классическими архитектурами с общей памятью. Поэтому, несмотря на то, что в чистом виде последовательные архитектуры микропроцессоров практически не встречаются, существенная доля приложений использует именно модель последовательного выполнения. При этом основным средством разработки программ для решения вычислительных задач являются традиционные языки программирования C, C++, FORTRAN.

2.2. Многопроцессорные системы с общей памятью

В системах с общей памятью предполагается наличие нескольких вычислительных ядер, которые имеют доступ к единому пространству памяти. Обмен данными между процессорами производится следующим образом: один процессор записывает данные по некоторому адресу, а другой считывает их. К этому классу относятся многопроцессорные рабочие станции и сервера, суперкомпьютеры с общей памятью (например HP Superdome), а также, получившие широкое распространение в последнее время многоядерные процессоры.

При доступе к общей памяти необходима *синхронизация* для обеспечения эксклюзивного доступа к данным. Различные механизмы синхронизации — мьютексы, семафоры, условные переменные предназначены для решения двух основных задач:

1. В то время как один поток выполняет операции с общими переменными, остальные потоки не могут выполнять эти операции до момента окончания выполнения операций этим потоком.
2. Как только один поток закончил выполнение операций с общими данными, один из ожидающих потоков получает доступ к этим данным.

Основным инструментарием разработчика программ для архитектур этого типа являются библиотека для организации многопоточных вычислений POSIX Threads[1] и пакет OpenMP[2]. Библиотека POSIX Threads имеет стандартизованный интерфейс и реализована для практически всех современных программно-аппаратных платформ. Пакет OpenMP основан на идее расширения традиционных языков программирования с помощью директив, управляющих параллелизмом. В настоящее время все наиболее распространенные трансляторы языков FORTRAN и C++ поддерживают OpenMP.

2.3. Многопроцессорные системы с распределенной памятью

Системы с общей памятью удобны для разработки параллельных программ и обеспечивают высокую производительность, но при большом

числе процессоров очень дороги. Менее затратной альтернативой являются системы с распределенной памятью. В таких системах каждый процессор имеет доступ только к своей локальной памяти, а между собой процессоры взаимодействуют с помощью передачи сообщений по сети. К этому классу относятся высокопроизводительные вычислительные кластеры, а также, локальные сети. Не редки также ЭВМ с гибридной архитектурой, в которой узлы с общей памятью соединяются посредством сетевого оборудования.

Основной парадигмой программирования для систем с распределенной памятью является передача сообщений. Наиболее распространенным средством разработки приложений в таких системах является MPI (Message Passing Interface) [3]. Библиотека MPI предоставляет набор функций для передачи сообщений между процессорами.

Распределенные системы

Для выполнения вычислений могут быть использованы не только специализированные многопроцессорные вычислительные комплексы, описанные в начале этого раздела, но и, в принципе любая совокупность вычислительных ресурсов, объединенных сетью [4]. При этом ресурсы могут располагаться в разных точках, обладать различной ведомственной принадлежностью, иметь разную архитектуру. Для доступа к таким ресурсам применяются различные механизмы. Некоторые узлы доступны через сетевой протокол прикладного уровня SSH. Ряд ресурсов может предоставляться с помощью различных программных средств промежуточного программного обеспечения Грид (gLite, UNICORE, Globus) в рамках Грид-ассоциаций. Мы будем в дальнейшем рассматривать *распределенные вычислительные системы*, состоящие из разнородных разноуровневых географически-распределенных вычислительных ресурсов с различными способами доступа.

Перечислим основные особенности распределенных систем, которые отличают их от параллельных систем с распределенной памятью:

- Неоднородный состав: вычислительные узлы могут существенно отличаться по архитекутуре, производительности, набору установленного программного обеспечения.
- Многообразие способов доступа к вычислительным узлам: для доступа могут применяться различные способы аутентификации и авторизации (secure shell, сертификаты X.509 и прочие способы доступа).
- Состав распределенной системы может изменяться по ходу вычислений.
- Относительно невысокая и неоднородная скорость передачи данных между узлами. Между некоторыми узлами распределенной системы возможно установление прямого соединения, в других случаях необходимо применять механизмы туннелирования или обмен через файлы.

Перечисленные различия между параллельными и распределенными системами служат причиной различий в подходах к численному решению задач на них. При организации распределенных вычислений необходимо применять методы динамической балансировки нагрузки и предусматривать возможность изменения состава распределенной среды в процессе вычислений и уметь синтезировать это пространство. Кроме этого, алгоритмы, предназначенные для использования в среде распределенных вычислений, должны допускать декомпозицию на совокупность слабо связанных между собой заданий, между которыми не происходит интенсивного обмена данными.

3. Принципы реализации алгоритмов оптимизации на различных программно-аппаратных платформах

Как следует из раздела 2, в настоящее время существует достаточно большое разнообразие платформ — от однопроцессорных систем до распределенных, содержащих десятки тысяч процессоров. Разработка программного обеспечения для решения задач оптимизации для каждой из этих платформ представляет собой достаточно сложную задачу. Если требуется решать m задач оптимизации на n типах вычислительных устройств, то, в общем случае, нужно создать $m \times n$ различных программ. Предлагаемый в данной работе подход позволяет сэкономить усилия разработчиков за счет повторного использования общих частей, имеющихся у методов решения различных задач. Фактически можно один раз реализовать общую схему решения для разных платформ, а в дальнейшем для каждого конкретного класса задач оптимизации реализовывать только проблемно-зависимые модули. После этого, программа для решения конкретной задачи на заданной платформе получается объединением проблемно-зависимых модулей для этой задачи и проблемно-независимых модулей, реализующих общую схему для выбранной платформы. В результате вместо $m \times n$ требуется разработать $m + n$ программных модулей. Далее рассмотрим принципы реализации алгоритмов решения задач оптимизации для различных платформ.

3.1. Последовательный вариант реализации

Реализация для последовательной (однопроцессорной) архитектуры имеет самостоятельное значение и может рассматриваться в качестве базового блока реализаций для многопроцессорных архитектур. Метод ветвей и границ реализуется в соответствии с общей схемой, рассмотренной в разделе 1:

1) выбрать из списка $\{X_i\}$ элемент X_i;

2) выбрать точку $x \in X \cap X_i$ и обновить рекорд $f_{k+1} = \min(f_k, f(x))$;

3) последовательно проверить выполнение правил отсева $\xi_1,\ldots,\xi_m$. Если хотя бы для одного из правил имеет место $\xi_j(X_i)=1$, то выполняется переход к шагу 1. Тем самым элемент X_i исключается из дальнейшего рассмотрения;
4) произвести декомпозицию множества X_i и получить семейство подмножеств $\Delta(X_i)=\left\{X_i^0,..,X_i^m\right\}$. Добавить полученные подмножества к списку $\left\{X_i\right\}$.

Производительность данного алгоритма во многом определяется способом хранения списка подмножеств $\left\{X_i\right\}$. Наиболее легко реализуемой является организация хранения на базе очереди, стека или другого стандартного контейнера, предполагающего хранение полных копий создаваемых подмножеств. Недостатками такого способа хранения являются существенный объем памяти, требуемый для хранения подмножеств, и дополнительные расходы на копирование при добавлении элементов в список. Альтернативным более экономным вариантом является организация хранения информации на базе *дерева ветвления*. В ряде случаев оправдано применение структур данных, ориентированных на конкретную задачу. Например, МВГ для задачи о ранце с одним ограничением может быть очень эффективно реализован на базе булева вектора[5].

3.2. Реализация для многопроцессорных архитектур с общей памятью

Наиболее простым вариантом реализации представляется подход, при котором потоки независимым образом выполняют операции 1–4 последовательного алгоритма, описанного выше. При этом, список $\left\{X_i\right\}$ является общим для разных потоков. Для предотвращения ошибок при доступе к списку необходима синхронизация, обеспечивающая эксклюзивный доступ потоков к списку.

Такая схема будет эффективно работать только при условии $t_1/N >> t_2$, где t_1 — суммарное время выполнения локальных операций 2–3, а t_2 — суммарное время выполнения операций, требующих синхронизации, и самой синхронизации. Опыт реализации показывает, что для многих задач, например для задачи о ранце, это соотношение не выполнено даже при небольшом (2–4) числе потоков. Более гибкая схема предполагает наличие локального списка у каждого из потоков дополнительно к общему списку. Обозначим список j-го потока через L_j. Поток j работает по следующей схеме:

1. Если $L_j \neq \varnothing$, то выбрать $X_i \in L_j$. В противном случае $\left(L_j = \varnothing\right)$ выполняется следующая последовательность действий:
   - получить эксклюзивный доступ к списку $\{X_i\}$;
   - если $\{X_i\} \neq \varnothing$, то выбрать из списка $\{X_i\}$ элемент X_i. В противном случае, встать в ожидание до момента появления в списке $\{X_i\}$ хотя бы одного элемента;
   - закончить эксклюзивный доступ к списку $\{X_i\}$.
2. Выбрать точку $x \in X \cap X_i$ и обновить рекорд $f_{k+1} = \min(f_k, f(x))$.
3. Последовательно проверить выполнение правил отсева $\xi_1,\ldots,\xi_m$. Если хотя бы для одного из правил имеет место $\xi_j(X_i) = 1$, то выполняется переход к шагу 1. Тем самым элемент X_i исключается из дальнейшего рассмотрения.
4. Произвести декомпозицию множества X_i и получить семейство подмножеств $\Delta(X_i) = \left\{X_i^0,..,X_i^m\right\}$. Добавить полученные подмножества к списку L_j.
5. Если число итераций кратно T, то добавить S элементов из списка L_j в список $\{X_i\}$.

Параметры T и S выбираются таким образом, чтобы обеспечивать заполнение списка $\{X_i\}$, не слишком часто прибегая к синхронизации. Для задач, обладающих высокой вычислительной сложностью, возможно переполнение общего списка $\{X_i\}$ за счет слишком интенсивного поступления подмножеств от потоков. Для предотвращения подобной ситуации вводятся два пороговых значения m_0 и M_0, $0 \le m_0 \le M_0$. Если число подмножеств в общем списке превосходит M_0, то процесс передачи подмножеств от потоков в общий список блокируется. В последующем, если количество подмножеств становится меньше m_0, то процесс передачи подмножеств возобновляется.

3.3. Реализация для многопроцессорных архитектур с распределенной памятью

Общая идея реализации МВГ на системах с распределенной памятью заключается в том, что каждый процессор работает со своим списком подмножеств, сохраняемым в локальной памяти. Если в результате выполне-

ния разбиений локальный список становится пустым, то для предотвращения простоев дополнительные множества пересылаются с другого процессора. Задача считается решенной, когда на всех процессорах списки не содержат подмножеств. В данной работе рассматриваются *централизованные* схемы организации вычислений: выделенный управляющий процессор отправляет рабочим процессорам области поиска и последний рекорд. Эти процессоры выполняют разбиения полученного подмножества, при необходимости пересылая на управляющий процессор часть подмножеств.

Рассмотрим два возможных способа параллельной реализации нахождения рекорда в методе ветвей и границ. При первом подходе разбиение и нахождение рекорда выполняются на каждом процессоре, который периодически переключается между этими действиями. Такой способ особенно удобен, когда для построения рекорда используются точки, полученные в процессе разбиения, к которым применяется один из методов локальной оптимизации. В тоже время в рамках данной схемы сложно создать параллельную реализацию алгоритма, работающего со всей совокупностью точек $x_1,\ldots,x_k$. В этом случае предпочтительнее схема, при которой одна группа процессоров выполняет разбиения, а другая задействована в поиске рекорда (рис. 2).

Рассмотрим базовую схему, подходящую для большинства алгоритмов, применяемых при нахождении рекорда. Выделенный управляющий процессор поддерживает список точек $x_1,\ldots,x_k$. Некоторая точка удаляется из списка и рассылается всем либо некоторым из рабочих процессоров, которые подвергают ее процедуре оптимизации (как правило, одному из вариантов локального поиска). В результате на рабочем процессоре образуется список точек, который пересылается управляющему процессору и объединяется с его списком. Максимальная длина списка точек, правила выбора очередной точки для обработки, распределение точек по рабочим процессорам, способ добавления точек, полученных от рабочих процессоров к общему списку, локальный алгоритм являются параметрами общей схемы и определяют конкретный эвристический метод поиска рекорда. Список точек на управляющем процессоре также может пополняться точками, полученными в результате разбиений.

На рис. 2 процессы помечены следующим образом: M — процесс, управляющий работой всего приложения; $\{B_i\}$ — процессы, которые выполняют операции разбиения, BM — процесс, координирующий работу процессов $\{B_i\}$, $\{H_i\}$ — процессы, которые выполняют эвристический поиск, HM координирует работу этих процессов. На начальном этапе процесс M рассылает исходные данные задачи остальным процессам. Разбиение выполняется группой, состоящей из управляющего процесса BM и рабочих процессов B_i. Процесс HM получает точки от процессов H_i и B_i, обновляет значение рекорда и рассылает его по всем процессам B_i, реали-

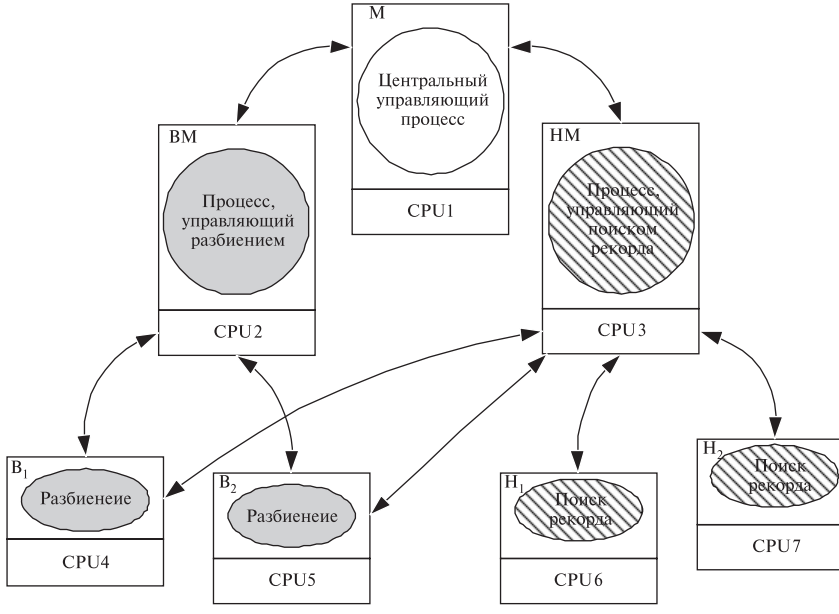

Рис. 2. Разбиение и построение рекорда выполняются на разных процессорах

зующим разбиения. Также НМ направляет полученную точку всем (или некоторым, в зависимости от выбранного алгоритма) процессам H_i, выполняющим эвристический поиск. Если на процессе H_i удается улучшить рекорд, то соответствующая рекордная точка посылается процессу НМ.

Применяется следующий простой и достаточно эффективный алгоритм балансировки, построенный на тех же принципах, что и алгоритм управления потоками в случае общей памяти. Процесс B_i управляется двумя параметрами — T и S, посылаемыми ему управляющим процессом. Получив подмножество для обработки, B_i выполняет T разбиений, после чего пересылает S сгенерированных подмножеств на управляющий процесс ВМ. По окончании разбиения своего подмножества процесс B_i получает от ВМ новое подмножество. Для предотвращения переполнения памяти на управляющем процессе было введено два пороговых параметра m_0 и M_0, $0 \le m_0 \le M_0$. Если число подмножеств на управляющем процессе становится больше M_0, то управляющий процесс рассылает всем рабочим процессам новое значения T, равное –1, и процессы B_i перестают пересылать подмножества управляющему процессу ВМ. Когда число подмножеств на управляющем процессе становится меньшим m_0, он рассылает рабочим процессам первоначальные значения T, S и передача подмножеств с рабочих процессов управляющему возобновляется.

3.4. Реализация в среде распределенных вычислений

Теоретически распределенная система может рассматриваться как вычислительный кластер очень большого размера. На практике особенности, перечисленные в подразделе «Распределенные системы» п. 2, не позволяют эффективно использовать распределенные системы при таком подходе.

В отличие от параллельных систем, распределенные системы имеют иерархическую организацию: они состоят из разнородных узлов, каждый из которых может, в свою очередь, быть параллельной системой, т. е. одной из систем, рассмотренных в предыдущих разделах. Естественно предположить, что максимальная эффективность достигается, когда вычислительный процесс организован в соответствии с этой иерархией. При таком подходе на каждом из узлов вычисления выполняются по наиболее подходящей для данного узла схеме. Например, если узел представляет собой многопроцессорную систему с распределенной памятью, то вычисления организуются по схеме, изложенной в подразделе «Реализация для многопроцессорных архитектур с распределенной памятью» п. 3. Фактически, на каждом из узлов распределенной системы выполняется отдельное приложение — *солвер*, выполняющее операции выбранного алгоритма оптимизации. Взаимодействие между несколькими приложениями организуется на следующем уровне иерархии через выделенный центральный управляющий процесс — *супервизор*.

Первый этап решения задачи в распределенной вычислительной среде состоит в *синтезе вычислительного пространства*, которое формируется экземплярами солверов. При большом числе узлов запуск приложений «вручную» может быть достаточно трудоемкой процедурой. Поэтому необходимо обеспечить возможность автоматизации запуска солверов супервизором. При этом используются средства удаленного доступа, предусмотренные конкретной системой: SSH, Грид-сервисы и др.

После создания вычислительное пространство может быть использовано для решения задачи. В процессе решения необходимо распределять вычисления между солверами с целью обеспечения максимальной эффективности выполнения приложения в распределенной среде. Обмен данными между солверами и супервизором осуществляется средствами, предусмотренными для взаимодействия с конкретным узлом. Если имеется возможность установления прямого сетевого соединения, то используются способы обмена, основанные на протоколах TCP/IP, например интерфейс сокетов. В некоторых случаях обмен данными с приложениями возможен только через передачу файлов средствами промежуточного программного обеспечения Грид.

Балансировка нагрузки производится на двух уровнях: на верхнем уровне супервизор распределяет вычислительную нагрузку между солве-

рами. На нижнем уровне (в пределах одного вычислительного узла) распределение работы производится солвером методами, предназначенными для конкретного типа вычислительного узла

4. Реализация

Предложенный подход был реализован в виде двух программных систем BNB-Solver и BNB-Grid. Система BNB-Solver предназначена для решения оптимизационных задач на однопроцессорных системах и суперкомпьютерах с распределенной и общей памятью. Система BNB-Grid позволяет решать задачи в распределенной вычислительной среде, в состав которой могут входить как суперкомпьютеры, так и однопроцессорные системы. В качестве компонентов, выполняющихся на узлах распределенной системы, используются экземпляры библиотеки BNB-Solver. Таким образом, вычисления организованы по иерархической двухуровневой схеме, соответствующей естественной иерархии распределенной системы.

4.1. Библиотека BNB-Solver

Основой объектно-ориентированной Си++ библиотеки BNB-Solver являются каркасные модули, реализующие общие схемы метода ветвей и границ, эвристических и гибридных алгоритмов для однопроцессорных систем, систем с общей памятью и кластерных систем с распределенной памятью. Каркасные модули являются параметризованными классами Си++. В качестве параметров в них подставляются классы, инкапсулирующие проблемно-зависимые детали конкретных алгоритмов и задач. Для организации параллельного выполнения на системах с общей памятью используется библиотека POSIX Threads, для систем с распределенной памятью — библиотека MPI (Message Passing Interface).

На данный момент разработаны классы для задачи о ранце [5], непрерывной оптимизации с ограничениями и без [6], задачи коммивояжера (Игнатьевым А. Л.)[7]. Результаты экспериментов [8] показывают, что для ряда задач BNB-Solver существенно обгоняет по производительности существующие свободно-распространяемые библиотеки.

4.2. Система организации распределенных вычислений BNB-Grid

Программный комплекс BNB-Grid предназначен для решения задач оптимизации на распределенных системах, состоящих из рабочих станций, небольших многопроцессорных комплексов, доступных в монопольном режиме, суперкомпьютеров коллективного доступа. BNB-Grid запускает и организует взаимодействие параллельных приложений, решающих

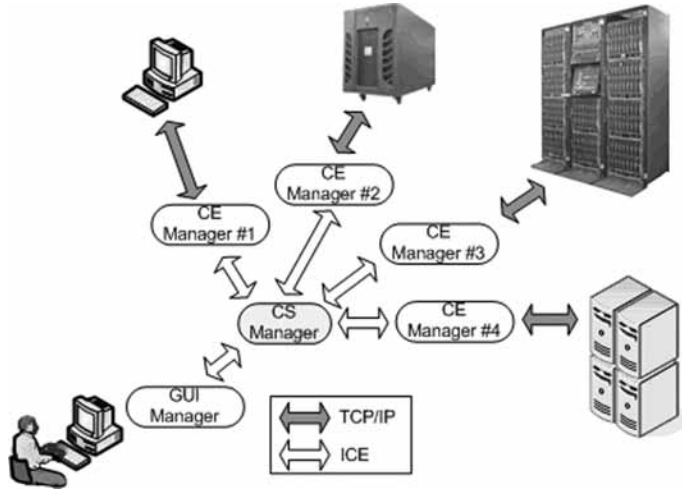

Рис. 3. Организация вычислений в BNB-Grid

задачу с помощью библиотеки BNB-Solver на вычислительных узлах. В результате формируется иерархическая распределенная система: на верхнем уровне части работы распределяются между параллельными приложениями, а далее они распределяются по процессорам средствами библиотеки BNB-Solver.

Ядро системы реализовано на языке программирования Java с использованием промежуточного программного обеспечения Internet Communication Engine (ICE)[9] — аналога CORBA. Каждый вычислительный узел представлен в системе распределенным экземпляром объекта типа CE-Manager — Computing Element Manager (рис. 3). Этот объект предоставляет интерфейс для запуска и остановки приложений на вычислительном узле. Координация работы объектов типа CE-Manager осуществляется с помощью другого распределенного объекта типа CS-Manager — Computing Space Manager. Графический интерфейс пользователя также реализован как ICE-объект, обозначенный на рисунке GUI Manager.

ICE-объекты размещаются либо на одном, либо на нескольких компьютерах в приделах локальной сети. Для обеспечения длительной автономной работы этих компонентов в фоновом режиме на сервере применяется технология ICE-Grid, автоматизирующая запуск сервисов ICE и унифицирующая доступ к удаленным объектам.

Доступ к удаленному вычислительному узлу осуществляется по представленной на рис. 4 схеме. Объект CE-Manager устанавливает SSH-соединение с внешним управляющим модулем узла, запускает на нем процесс CE-Server и устанавливает с ним TCP/IP соединение. Если удален-

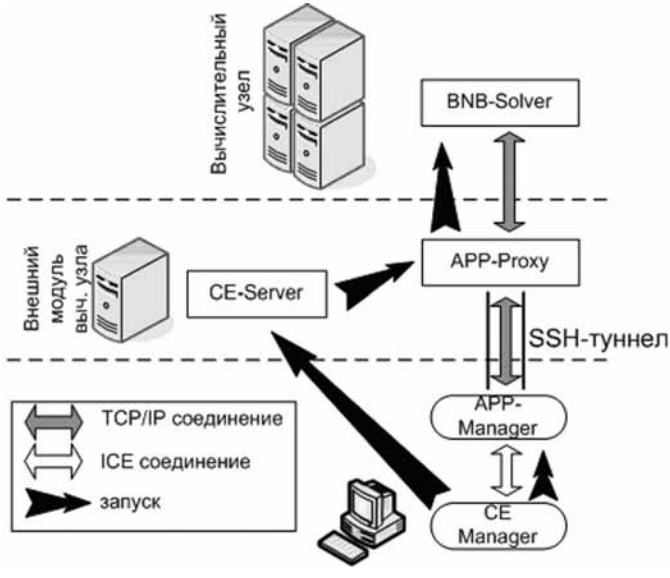

Рис. 4. Запуск приложений на удаленном вычислительном узле и установление соединения

ный узел защищен сетевыми экранами, то применяется механизм туннелирования сетевого соединения через SSH-канал. Процесс CE-Server информирует CE-Manager о состоянии вычислительного узла. При обрыве соединения или перезагрузке узла CE-Server перезапускается. При получении запроса на запуск параллельного приложения CE-Manager создает объект APP-Manager, CE-Server создает процесс APP-Proxy, который запускает параллельное приложение BNB-Solver на узле. При этом CS-Manager взаимодействует с APP-Manager средствами ICE, а APP-Manager, в свою очередь, устанавливает TCP/IP соединение с BNB-Solver через APP-Proxy. Процесс APP-Proxy необходим, так как на суперкомпьютерах общего доступа непосредственный доступ из внешней сети к вычислительным модулям, как правило, невозможен.

На одном вычислительном узле может быть запущено несколько экземпляров приложения BNB-Solver. Такой подход часто оказывается целесообразным для суперкомпьютеров коллективного доступа, работающих под управлением систем пакетной обработки. На таких системах приложение, запросившее достаточно большое число процессоров, может быть надолго помещено в очередь в ожидании соответствующего «окна» в расписании заданий. В то же время, приложение, запросившее существенно меньшее число процессоров, может быть запущено ранее. Этот эффект

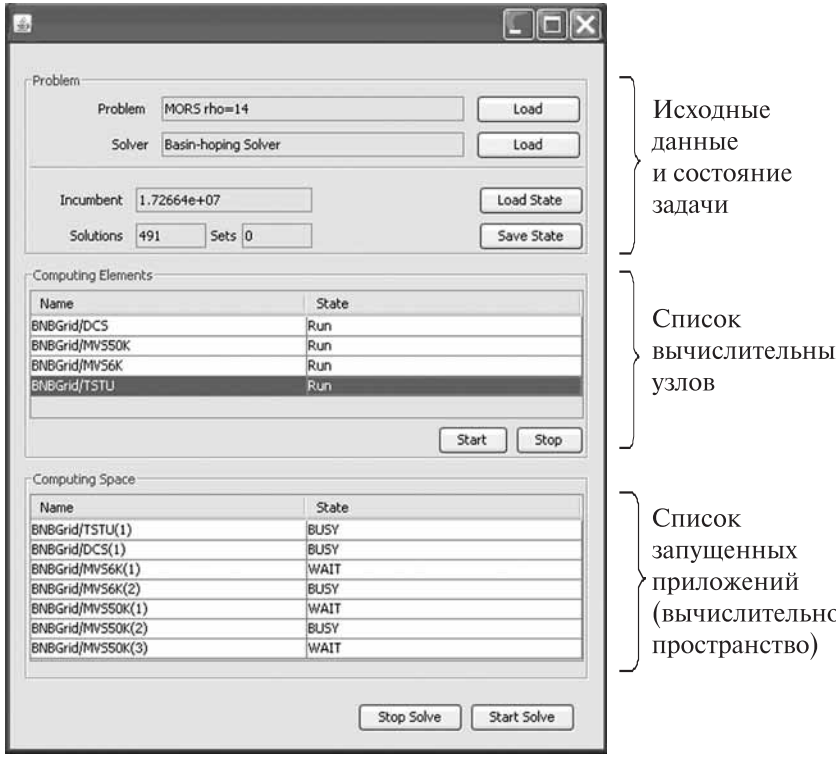

Рис. 5. Графический интерфейс пользователя системы BNB-Grid

объясняется тем, что системе пакетной обработки проще эффективно размещать небольшие задания. Поэтому в наших экспериментах на суперкомпьютерах коллективного доступа запускалось от пяти до двадцати приложений.

Выделение графического интерфейса пользователя в отдельный ICE-объект GUI-Manager преследует две цели. Во-первых, для работы в системе BNB-Grid пользователю достаточно установить на своем персональном компьютере только этот компонент. Во-вторых, при таком подходе графический интерфейс становится легко заменяемым компонентом, который не является неотъемлемой частью системы, взаимодействуя с помощью ICE. Компонент GUI-Manager служит для инициализации вычислений, управления и мониторинга. Он может устанавливать и разрывать соединение с компонентом CS-Manager в произвольные моменты времени. Таким образом, вычисления могут производиться автономно без участия оператора достаточно длительное время.

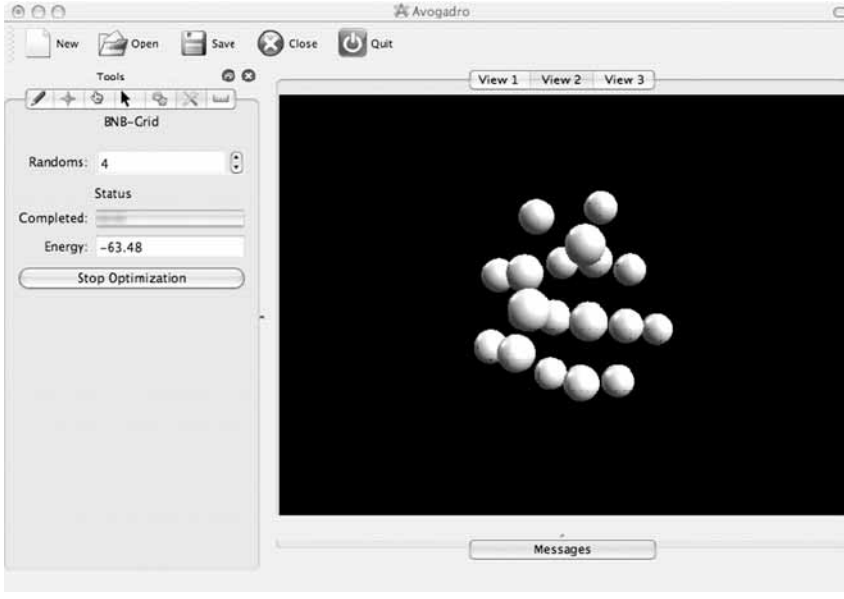

Рис. 6. Версия графического интерфейса пользователя системы BNB-Grid для решения задачи поиска энергетически-минимальных конформаций молекулярных кластеров

В качестве примера можно рассмотреть общий (рис. 5) и специализированный для задачи поиска энергетически-оптимальной конформации молекулярного кластера (рис. 6) варианты графического интерфейса. Общий вариант предоставляет возможности для загрузки исходных данных задачи, управления вычислительным пространством, загрузки и сохранения состояния решения задачи. Специализированный графический интерфейс, разработанный Смирновым С. А., позволяет просматривать и модифицировать исходные и полученные в результате расчетов кластеры. Вместе с системой хранения конформаций этот интерфейс образует развитую оболочку для исследования данной задачи.

Система BNB-Grid была успешно применена для решения задачи поиска энергетически-минимальной конформации молекулярного кластера[10]. Результаты экспериментов показали, что суммарная вычислительная мощность нескольких суперкомпьютеров позволяет существенно раздвинуть границы применимости этого метода и получать оптимальные либо близкие к оптимальным конформации за приемлемое время для различных потенциалов межатомного взаимодействия.

Заключение

Предложены методы и основанная на них двухуровневая иерархическая программная инфраструктура для решения комбинаторных и оптимизационных задач большой размерности в среде распределенных и параллельных вычислений. Эта инфраструктура позволяет решать задачи оптимизации на последовательных, параллельных системах с общей и распределенной памятью, Грид-системах. Проблемно-зависимая часть отделена от общей схемы метода, что позволяет существенно уменьшить усилия, необходимые при реализации новой оптимизационной задачи. Разработанный подход является теоретической базой для создания программного обеспечения. Результаты апробации подхода для различных задач дискретной и непрерывной оптимизации показали его высокую эффективность.

Литература

1. *Drepper U., Molnar I.* The Native POSIX Thread Library for Linux. 2005: http://people.redhat.com/drepper/nptl-design.pdf
2. *Chandra R.*, *Dagum L.*, *Kohr D.*, *Maydan D.*, *McDonald J.*, *Menon R.* Parallel Programming in OpenMP Morgan Kaufmann. 2001. 248 p.
3. *Snir M.*, *Otto S.*, *Huss-Lederman S.*, *Walker D.*, *Dongarra J.* MPI: The Complete Reference. MIT Press. Boston. 1996.
4. *Emelyanov S. V.*, *Afanasiev A. P.*, *Grinberg Y. R.*, *Krivtsov V. Y.*, *Peltsverger B. V.*, *Sukhoroslov O. V.*, *Taylor R. G.*, *Voloshinov V. V.* Distributed Computing and Its Applications. Editor: Velikhov, E. P. Bristol, ME, USA: Felicity Press. 2005.
5. *Посыпкин М. А., Сигал И. Х.* Комбинированный параллельный алгоритм решения задачи о ранце // Известия РАН. Теория и системы управления. 2008. № 4. С. 50–58.
6. *Посыпкин М. А.* Решение задач математического программирования на многопроцессорных вычислительных системах // Труды Третьей Международной научно-практической конференции «Современные информационные технологии и ИТ-образование» Москва, 6–9 декабря 2008 г. М.: МАКС ПРЕСС, 2008. С. 490–499.
7. *Игнатьев А. Л.* Сравнение различных методов решения задачи коммивояжера на многопроцессорных системах // Труды Третьей международной научно-практической конференции «Современные информационные технологии и ИТ-образование» Москва, 6–9 декабря 2008 г. М.: МАКС ПРЕСС, 2008. С. 509–514.
8. *Evtushenko Y., Posypkin M., Sigal I.* A framework for parallel large-scale global optimization // Computer Science — Research and Development 23(3). 2009. P. 211–215.
9. *Henning M.* A New Approach to Object-Oriented Middleware // IEEE Internet Computing, Jan 2004.
10. *Посыпкин М. А.* Методы и распределенная программная инфраструктура для численного решения задачи поиска молекулярных кластеров с минимальной энергией // Параллельные вычислительные технологии (ПаВТ’2009): Труды международной научной конференции (Нижний Новгород, 30 марта – 3 апреля 2009 г.). Челябинск: ЮУрГУ, 2009. С. 269–281.