МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Н.В. Старостин
М.А. Быкова

МНОГОУРОВНЕВЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ АРХИТЕКТУРНО-ЗАВИСИМОЙ ДЕКОМПОЗИЦИИ

Учебно-методическое пособие

Рекомендовано методической комиссией ИИТММ
для студентов ННГУ, обучающихся по направлениям подготовки
09.04.03 «Прикладная информатика» и
01.03.02 «Прикладная математика и информатика»

Нижний Новгород
2017

УДК 519.687.6
ББК 22.1я7

Н.В. Старостин, М.А. Быкова, МНОГОУРОВНЕВЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ АРХИТЕКТУРНО-ЗАВИСИМОЙ ДЕКОМПОЗИЦИИ. Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2017. – 24 с.

Рецензент: д.т.н., профессор В.Е. Турлапов

В данном учебно-методическом пособии излагается постановка и алгоритм решения задачи архитектурно-зависимой декомпозиции.

Пособие предназначено для бакалавров и магистров направлений подготовки «Прикладная информатика» и «Прикладная математика и информатика».

УДК 519.687.6
ББК 22.1я7

Оглавление

1. Задача архитектурно-зависимой декомпозиции

Современные вычислительные системы – это разделяемый ресурс, на котором одновременно могут выполняться несколько заданий (независимых процессов). Для эффективного использования требуется решать задачи, связанные с планированием и запуском заданий. Планированием и запуском параллельных задач занимается специальная служба операционной системы (планировщик). Проблемы планирования заданий в высокопроизводительной среде можно разделить на два класса — планирование независимых задач и планирование взаимодействующих задач. Планирование независимых задач заключается в сбалансированном распределении задач по вычислителям, в то время как при планировании взаимодействующих задач необходимо распределять задачи с учетом их информационных обменов, чтобы минимизировать стоимость межпроцессорных коммуникаций. Допустим, планировщик выделил некоторой программе (заданию) требуемое число вычислителей. Очевидно, что время выполнения программы будет зависеть от физической топологии выделенного участка вычислительной сети и от распределения параллельных подзадач по вычислителям, поскольку время передачи данных в каждом случае будет разным (интенсивно обменивающиеся данными подзадачи могут оказаться «далеко» друг от друга в рамках физической топологии вычислительной сети). Возникает необходимость адаптировать физическую топологию фрагмента вычислительной сети, которая досталась программе, к той топологии, которая наиболее для нее оптимальна с точки зрения коммуникационных затрат. То есть необходимо решить задачу оптимального отображения параллельных подзадач на физическую топологию вычислительной сети.

В работе рассматривается проблема планирования взаимодействующих задач, т.е. частей одной параллельной задачи с целью ускорения ее выполнения на определенной вычислительной системе. Предполагается, что распараллеливание программы выполнено заранее и изменению не подлежит. В работе приводится формализация описанной проблемы, рассматриваются ее частные случаи, которые могут быть сведены к известным оптимизационным задачам, проводится исследование поставленной задачи, предлагается многоуровневый алгоритм ее решения.

Далее в работе все задачи, исходными данными которых являются параллельная задача (расчетная сетка, граф зависимостей параллельной программы), параллельная вычислительная система (матрица, описывающая ее свойства, граф вычислительной системы) и необходимо распределить части параллельной программы по вычислителям с некоторыми ограничениями и критериями (минимизация дисбаланса, минимизация внешних связей) будем называть задачами архитектурно-зависимой декомпозиции.

1.1. Содержательная постановка задачи

Традиционно задачу планирования рассматривают как многостадийный процесс, который осуществляет построение и отслеживание очереди задач, эффективное распределение ресурсов вычислительной системы по задачам, запуск задач. Рассмотрим все этапы планирования задач в параллельной вычислительной системе. Обычно эту функцию выполняет специальная служба операционной системы - планировщик. Задачи, ожидающие выполнения, формируют очередь. Каждой задаче требуется выделить заданное количество вычислителей на заданное время. Планировщик извлекает задачи из очереди по одной или пачками и назначает их на свободные вычислители. Очевидно, что для более эффективного использования ресурсов параллельной вычислительной системы каждой задаче необходимо выделять «близкие» с точки зрения физической топологии системы вычислители. Это позволит экономить время во время пересылок информации между параллельными частями задачи, выполняющимися на разных вычислителях. На описанном этапе решается задача построения расписания.

Параллельные задачи, на которых сконцентрировано исследование, представляют собой расчетные сетки больших размеров. Как правило, размер сетки много превышает количество доступных вычислителей для ее обработки, а также данные задачи могут не вместиться в память одного вычислителя. Поэтому, перед началом выполнения параллельной программы, необходимо произвести ее декомпозицию на требуемое количество вычислителей и загрузить часть декомпозированной сетки на каждый вычислитель, выделенный для ее выполнения.

После того, как планировщик выделил необходимое количество вычислителей для очередной задачи, и была произведена ее декомпозиция и загрузка на вычислители, можно запускать процесс выполнения этой задачи. Однако, можно попытаться еще увеличить эффективность использования вычислительной системы и сократить время, необходимое для пересылки данных между вычислителями, а также время простоя отдельных вычислителей.

Время выполнения параллельной программы зависит не только от ее сложности и мощности вычислителей. На данные характеристики повлиять практически невозможно. Однако, можно повлиять на эффективность использования вычислительной системы, просто правильно распределив параллельные части программы по вычислителям. Выделим две характеристики, от которых зависит время выполнения программы, а, следовательно, и время использования дорогостоящей техники. К этим характеристикам относятся сбалансированность загрузки вычислителей и количество данных, которые необходимо пересылать между вычислителями в процессе выполнения программы. Несбалансированная загрузка вычислителей приводит к простою отдельных вычислителей, а большое количество

межпроцессорных коммуникаций может занять существенное время. Поэтому необходимо минимизировать дисбаланс загрузки вычислителей и количество межпроцессорных коммуникаций. На этих проблемах и сконцентрирована работа.

В работе предлагается автоматизация процесса перераспределения частей параллельной программы по выделенным вычислителям с целью минимизации дисбаланса загрузки вычислителей и межпроцессорных коммуникаций. Этот процесс выглядит следующим образом: задача сообщает планировщику о своей структуре, планировщик, зная о физической топологии участка вычислительной сети, выделенного для выполнения этой задачи, распределяет ее части по вычислителям с учетом указанных критериев.

Исходными данными рассматриваемой задачи являются параллельная программа и вычислительная сеть (участок вычислительной сети), выделенная для выполнения программы. Параллельная программа характеризуется количеством параллельных частей, вычислительной сложностью каждой параллельной части, а также структурой и интенсивностью коммуникационных обменов между параллельными частями. Вычислительная сеть характеризуется количеством вычислителей, их вычислительной мощностью, а также структурой коммуникационной топологии. Под структурой коммуникационной топологии вычислительной сети понимается расположение коммуникаторов, физических связей между коммуникаторами и вычислителями, коммуникационных шин между вычислителями, а также коммуникационные характеристики коммуникаторов и шин (стоимость передачи данных). Особенностями исходных данных являются большой размер параллельной программы (количество неизвестных в задаче достигает 10^6-10^9), необходимость распределенного хранения данных программы, разреженный неструктурированный характер коммуникаций между параллельными частями программы.

В задаче необходимо распределить параллельные части программы по вычислителям (т.е. каждой части программы поставить в соответствие ровно один вычислитель) так, чтобы вычислители были загружены одинаково с учетом разной сложности частей программы и разной мощности вычислителей (условие балансировки), а суммарная стоимость передачи информации между частями программы в процессе выполнения была минимальной (коммуникационное условие).

Очевидно, что далеко не всегда возможно сбалансированно разделить параллельные части программы по вычислителям. Простым примером может послужить программа, состоящая из двух параллельных частей разной вычислительной сложности и вычислительная система, состоящая из двух вычислителей одинаковой мощности. Таким образом, условие балансировки может быть сформулировано в виде ограничения с заранее заданным допустимым отклонением от идеально сбалансированного разбиения или в виде критерия, минимизирующего это отклонение. Коммуникационное условие как

правило формулируется в виде критерия, минимизирующего суммарную стоимость коммуникаций (аддитивный критерий) или максимальную стоимость коммуникаций (минимаксный критерий).

Решение такой задачи позволит более эффективно использовать вычислительную сеть и ускорить выполнение параллельной программы. Впоследствии все задачи, имеющие подобную содержательную постановку, будем называть задачами архитектурно-зависимой декомпозиции.

1.2. Математическая постановка задачи

В основу математической модели исходных данных параллельной программы предлагается положить неориентированный взвешенный граф $G(V,E,c,u)$, где $V=\{v_1,...,v_n\}$ – множество вершин, моделирующее данные параллельной программы, каждая вершина представляет неделимую единицу данных, которую можно передать на любой вычислитель для выполнения расчета, $E \subseteq V^{(2)}$ – множество ребер, моделирующее зависимости по данным между исходными данными программы; веса вершин $c: V \to N$ – моделируют относительные вычислительные издержки на выполнение расчета определенной единицы данных программы; веса ребер $u: E \to N$ моделируют объём коммуникаций между единицами данных параллельной программы.

В основу математической модели вычислительной системы предлагается положить взвешенный гиперграф $H(Y,A,p,w)$, где $Y=\{y_1,...,y_k\}$ – множество вершин, моделирующих вычислители системы; $A \subseteq 2^Y$ – множество гиперребер, моделирующих коммуникационные связи между вычислителями (в некоторых случаях коммуникационные каналы представляют собой программно-аппаратные реализации разделяемых шин, в рамках которых осуществляется передача данных между вычислителями (примером может послужить архитектура связей ядер многоядерного процессора). В таких случаях множество вершин гиперграфа, соответствующих этим вычислителям, образуют гиперребро); веса вершин $p: Y \to (0,1]$ – моделируют относительную производительность вычислителей; веса гиперребер $w: A \to N$ – определяют скорость пересылки данных (в общем случае может быть вектор характеристик коммуникационной среды (способ передачи информации (сообщениями или пакетами), время подготовки информации к отправке, скорость передачи единицы информации, время передачи служебной информации и т.п.)).

Сложности моделирования выполнения параллельной программы на высокопроизводительной вычислительной системе связаны с тем, что на практике оказывается очень сложно рассчитать маршрут передачи данных, совпадающий с маршрутом, выбранным маршрутизатором, учесть коллизии, возникающие во время передачи данных, задержки на сетевых устройствах. По этой причине приходится пользоваться оценками искомого времени,

основанными на кратчайших маршрутах передачи данных без учета коллизий. Модель зависимости оценки времени передачи данных между процессорами включает маршрут передачи данных, физические параметры каналов вычислительной сети, связанные с пропускной способностью, скоростью, латентностью, технологией передачи данных. В работе описаны трудности, связанные с явным выражением вида функции зависимости оценки времени передачи данных для современных высокопроизводительных вычислительных систем. Предлагается матричная аппроксимация гиперграфовой модели вычислительной системы, основанная на приближенном вычислении функции времени пересылки данных между парой процессоров.

В основу матричной модели вычислительной системы на k вычислителях положена матрица $S = \{s_{ij}\}_{k \times k}$. Элементы матрицы $s_{ij} \in N, i, j = \overline{1,k}$ являются коэффициентами линейной аппроксимации (интерполяции) функции временных затрат на пересылку от объема пересылаемых данных. В этом случае функция времени передачи данных объема m между парой процессоров i и j будет иметь вид: $T(i,j,m)=s_{ij}m$.

Удобство описанной модели обосновывается низкими издержками, связанными с расчетом временных затрат на пересылки данных определенного объема между парой вычислителей.

Решением задачи является матрица $X = \{x_{ij}\}_{n \times k}$, где элемент $x_{ij} = \begin{cases} 1, \text{если вершина } i \text{ назначена на вычислитель } j; \\ 0, \text{в противном случае} \end{cases}$.

Естественным ограничением на решение является требование, чтобы каждая вершина графа данных программы была назначена ровно на один вычислитель:

$$\sum_{j=1}^{k} x_{ij} = 1, i = \overline{1,n} \quad (1)$$

Равномерность загрузки с технической точки зрения означает, что все процессоры выполняют расчет на распределенных данных параллельно и заканчивают вычисления одновременно в рамках некоторой фазы. В случае, если какие-либо процессоры выполняют расчет дольше, то для синхронной схемы организации параллельного расчета это означает простой остальных процессоров. В этом случае решение поставленной задачи должно отвечать ограничению баланса, чтобы вычислители были загружены равномерно. Рассчитаем вычислительную нагрузку на каждый вычислитель:

$$L_j = \sum_{i=1}^{n} x_{ij} \cdot c_i, \; j = \overline{1,k} \quad (2)$$

Рассчитаем идеальную вычислительную нагрузку на каждый вычислитель, обеспечивающую одновременное завершение работы всех вычислителей:

$$\tilde{L}_j = p_j \sum_{i=1}^{n} c_i,\ j = \overline{1,k} \qquad (3)$$

Введем действительный параметр $\varepsilon > 0$, который определяет величину допустимого дисбаланса в загрузке вычислителей с учетом их производительности.

Потребуем, чтобы максимальное отклонение загрузки от идеального не превысило заранее заданного порога ε:

$$\max_{j=\overline{1,k}} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \le \varepsilon, \varepsilon \ge 0 \qquad (4)$$

При $\varepsilon = 0$ получаем требование идеально сбалансированного разбиения.

Для случая асинхронной модели организации параллельного расчета важнее оценивать среднее отклонение загрузки процессоров от идеального:

$$\frac{1}{k} \sum_{j=1}^{k} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \le \varepsilon, \varepsilon \ge 0 \qquad (5)$$

При $\varepsilon = 0$ получаем требование идеально сбалансированного разбиения.

Отметим, что если формулировать требование сбалансированного распределения данных параллельной программы по вычислителям в виде ограничений, то задача выбора допустимого решения *X*, удовлетворяющего ограничению (4) или (5), является NP-полной (частный случай задачи о камнях (разбиения)), и решение данной проблемы сводится к комбинаторному перебору всех вариантов, что с практической точки зрения не представляется возможным. С другой стороны не исключена ситуация, в которой система ограничений будет несовместна. С этой точки зрения практический интерес представляют постановки задач, в которых обозначенные ограничения (4) и (5) сформулированы в виде критериев.

Минимаксный критерий (6) и аддитивный критерий (7) нацелены на минимизацию дисбаланса относительной загрузки вычислителей:

$$\max_{j=\overline{1,k}} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \to \min \qquad (6)$$

$$\sum_{j=1}^{k} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \to \min \qquad (7)$$

Помимо требований, связанных с балансом, общее время работы параллельной программы определяется временными затратами на межпроцессорные пересылки. Как правило межпроцессорные коммуникации в параллельной программе возникают между теми процессорами, на которые распределены зависимые между собой данные программы, которые описываются графовой моделью. На практике очень сложно, а зачастую практически невозможно спрогнозировать: точное время каждой коммуникации; возникновение коллизий на уровне сетевых устройств;

снижение пропускной способности каналов в связи с повышением интенсивности обменов; маршрутов, по которым происходит передача пакетов. Все это значительно снижает адекватность применения модели в терминах планирования коммуникационных обменов. С практической точки зрения удобнее оперировать оценками затрат на всю совокупность коммуникационных обменов.

В качестве оценок затрат на межпроцессорные коммуникации параллельной программы будем использовать интервальную модель. Случай, когда все обмены происходят одновременно и не конфликтуют, означает, что достигается нижняя оценка времени выполнения программы. Потребуем минимизации этой оценки в критерии (8):

$$F_1(x) = \max_{i,j=\overline{1,k}} s_{ij} \sum_{l=1}^{n}\sum_{d=1}^{n} x_{li} \cdot x_{dj} \cdot u_{ld} \to \min . \quad (8)$$

Случай, когда все обмены конфликтуют друг с другом и происходят последовательно, является верхней оценкой. В данном случае имеет место аддитивный критерий (9):

$$F_2(x) = \sum_{i=1}^{k}\sum_{i=1}^{k} s_{ij} \sum_{l=1}^{n}\sum_{d=1}^{n} x_{li} \cdot x_{dj} \cdot u_{ld} \to \min . \quad (9)$$

Будем рассматривать задачу архитектурно-зависимой декомпозиции с ограничениями (1), (5) и критериями (7) и (9) в лексикографии (9, 7). Приоритет коммуникационного критерия объясняется тем, что балансировка нагрузки вычислителей учитывается в ограничении.

Итак, запишем полученную математическую постановку задачи:

$$\sum_{j=1}^{k} x_{ij} = 1, i = \overline{1,n} \quad (1)$$

$$\max_{j=\overline{1,k}} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \le \varepsilon_1, \varepsilon_1 \ge 0 \quad (5)$$

$$F_2(x) = \sum_{i=1}^{k}\sum_{i=1}^{k} s_{ij} \sum_{l=1}^{n}\sum_{d=1}^{n} x_{li} \cdot x_{dj} \cdot u_{ld} \to \min \quad (9)$$

$$\sum_{j=1}^{k} \left| \frac{L_j}{\tilde{L}_j} - 1 \right| \to \min \quad (7)$$

Среди частных случаев задачи следует выделить следующие:

1. Задача декомпозиции графа [1]: граф данных – произвольный, граф топологии – полный граф с вершинами одного веса и ребрами одного веса;
2. Квадратичная задача о назначениях [2]: граф данных и граф топологии имеют одинаковые размеры, вершины каждого графа имеют равные веса.
3. Задача коммивояжера [1]: граф данных – кольцо, граф топологии – произвольный;

4. Задача переупорядочения вершин графа с целью минимизации ширины ленты матрицы смежности [3]: граф данных – произвольный, граф топологии – линейка.

2. Многоуровневый алгоритм решения задачи архитектурно-зависимой декомпозиции

Основная идея предлагаемого алгоритма состоит в последовательном упрощении исходной задачи посредством отказа от некоторых исходных данных. Так на первом этапе (см. рис. 1) предлагается отказаться от информации о производительности процессоров и стоимости передачи данных между ними, что позволяет рассматривать задачу в терминах равномерного распределения вершин графа данных по процессорам. Формально такая задача рассматривается как сбалансированное k-разбиение графа данных параллельной программы, где k – количество процессоров. Решение этой частной задачи группирует вершины графа параллельной программы по подграфам. Такая группировка минимизирует связи между подграфами разбиения и тем самым локализует интенсивные обмены данными в рамках одного процессора. На втором этапе восстанавливается информация о стоимости передачи данных между процессорами, и решается задача назначения подграфов разбиения на процессоры, которая формулируется в виде квадратичной задачи о назначениях (см. рис. 1). При этом алгоритмы решения квадратичной задачи о назначениях пытаются отобразить связанные друг с другом подграфы на "близкие" процессоры в рамках физической топологии вычислительной сети, тем самым снижая издержки на межпроцессорные коммуникации. На первом и втором этапах не учитывался такой аспект вычислительных систем, как различная производительность вычислительных узлов. В результате синтезированное решение задачи отображения графа может не удовлетворять ограничению (5), и на третьем этапе происходит полное восстановление информации о характеристиках вычислительной системы (см. рис. 1), что позволяет провести локальную оптимизацию найденного решения задачи архитектурно-зависимой декомпозиции.

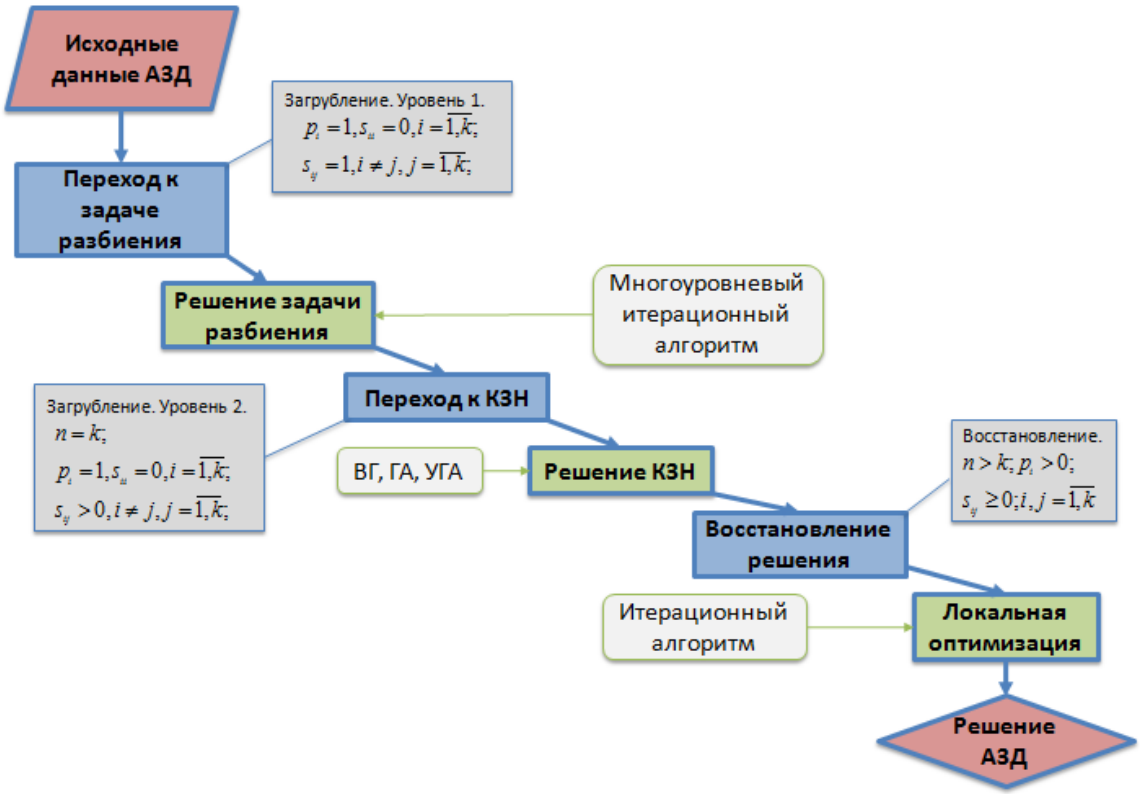

Рисунок 1. Схема многоуровневого алгоритма

Рассмотрим подробнее все этапы алгоритма.

2.1. Редукция задачи АЗД к квадратичной задаче о назначениях на основе k-разбиения графа

Переход от исходной задачи к задаче декомпозиции заключается в релаксации исходных данных – все вычислители считаются равномощными и стоимость передачи данных между каждой парой вычислителей считается одинаковой. Тогда задача архитектурно-зависимой декомпозиции вырождается в задачу декомпозиции графа параллельной программы на количество частей, равных количеству вычислителей системы. Такой прием позволяет сгруппировать интенсивно обменивающиеся данными части параллельной программы и уменьшить вес связей между подграфами декомпозиции. В последствии каждый подграф будет назначен на свой вычислитель, и интенсивные обмены данными будут сконцентрированы внутри вычислителей.

Для решения задачи декомпозиции графа будем использовать многоуровневый итерационный алгоритм [4]. Авторами предлагается концепция многоуровневого итерационного алгоритма, который многократно редуцируя и восстанавливая задачу и ее решение, позволяет на разных уровнях редукции первоначально полученное решение.

Задача редукции состоит в получении нового графа меньшего порядка. Главная идея заключается в том, чтобы отождествить те ребра, которые с высоким шансом попадают в один подграф оптимального разбиения. Как показала практика, высокие шансы имеют те ребра, которые топологически находятся дальше от сечения имеющегося разбиения. Оператор редукции случайно выбирает заданное число независимых ребер, из которых затем оставляет половину наиболее удаленных от сечения. Затем имеющееся решение формируется в терминах нового графа, локально улучшается, а затем восстанавливается разбиение в терминах исходного графа. В основе локальной оптимизации положен известный алгоритм Fiduccia-Mattheyses [5].

2.2. Схема построения квадратичной задачи о назначениях

По полученному разбиению строится редуцированный граф, где каждая вершина ассоциируется с подграфом декомпозиции. Ребро между двумя вершинами существует в том случае, если в исходном графе есть ребра, связывающие соответствующие подграфы. При этом вес данного ребра определяется как сумма весов этих ребер. Число вершин построенного графа совпадает с количеством процессоров. На этом этапе восстанавливается исходная информация о стоимости передачи данных между процессорами, однако различия в производительности вычислительных узлов по-прежнему не учитываются. Теперь необходимо распределить вершины полученного графа по вычислителям системы. По имеющимся данным предлагается построить квадратичную задачу о назначениях в матричной форме.

Матрица $A = (a_{ij})_{k \times k}$ ассоциируется с интенсивностью коммуникаций между ветками параллельной программы, матрица $S = (s_{ij})_{k \times k}$ описывает затраты на межпроцессорные коммуникации.

Решением задачи является перестановка p, которая ставит соответствие i-ой части программы процессор p(i). Целью задачи является минимизация суммарных издержек на пересылку данных.

$$\sum_{i=1}^{k} \sum_{j=1}^{k} a_{ij} s_{p(i)p(j)} \to \min \qquad (10)$$

Первую матрицу построим на основе весов ребер между вершинами полученного редуцированного графа. Вторая матрица хранит информацию о скорости передачи данных между каждой парой вычислителей. Для решения квадратичной задачи о назначениях предлагается генетический алгоритм, описанный ниже.

2.3. Генетический алгоритм решения квадратичной задачи о назначениях

Генетический метод представляет собой подход к решению оптимизационных задач, основанный на математическом моделировании эволюционных процессов (скрещивания, кроссовера, мутации, естественного отбора) [22]. Данный метод используется в работе для разработки алгоритмов решения квадратичной задачи о назначениях.

Основными этапами генетического алгоритма, используемого в работе являются: генерация стартового поколения, оценивание, выбор родительских пар, скрещивание, мутация, оценивание и отбор, проверка условия останова, генерация следующего поколения [22].

Параметрами генетического алгоритма являются: размер поколения (количество особей в поколении), количество родительских пар, участвующих в воспроизводстве особей, количество мутантов (количество особей, подвергающихся мутации), вероятность мутации определенной особи, условие останова [22].

Опишем подробнее все этапы и параметры генетического алгоритма для решения поставленной задачи.

2.3.1. Представление решений, функция приспособленности и генерация стартового поколения

В предлагаемом алгоритме используется порядковое кодирование особей [22]. Каждая особь является перестановкой, моделирующей решение задачи. Функцией приспособленности особи является значение критерия (1) на перестановке, кодирующей данную особь. Генерация начальной популяции в предлагаемом алгоритме происходит случайным образом (генерируется заранее заданное количество случайных перестановок). Такая стратегия позволяет сгенерировать решения, далекие друг от друга с точки зрения расстояния Хэмминга [22 и избежать преждевременной сходимости в локальный оптимум.

2.3.2. Репродукция решений

Для корректного функционирования генетического поиска необходимо генерировать новые решения на основе уже существующих. Это позволяет как сохранять качества родительских особей, так и вносить новые. Данный процесс называется репродукцией решений и является имитацией процесса размножения в природе.

За репродукцию решений отвечают операторы скрещивания и кроссовера. Оператор скрещивания отвечает за формирование родительских пар особей. Системой скрещивания, используемой в работе является панмиксия, в которой любые две особи имеют одинаковую возможность образовать родительскую пару [6]. Кроссовер осуществляет обмен генами или частями хромосом между двумя родителями с целью получения новых хромосом, являющихся кодировками их потомков.

Сложность операторов скрещивания для порядкового представления решений в отличии от бинарного представления заключается в необходимости следить за допустимостью решения, получаемого в результате скрещивания. Кроссовер для порядкового представления решений должен гарантировать сохранение перестановки. Примерами операторов скрещивания могут быть одноточечный, двухточечный, многоточечный, однородный кроссоверы для порядкового представления решений [6].

Опишем оператор скрещивания, разработанный для предлагаемого генетического алгоритма. Данный оператор нацелен на сохранение лучших качеств родительских особей в потомке. Для гарантии сохранения перестановки предлагаемый кроссовер конструирует потомка из прямой перестановки. Оператор сравнивает стоимость назначения элементов в очередной позиции родительских особей, и ставит на соответствующее место в дочернего решения элемент с меньшей стоимостью путем парного обмена этого элемента с элементом, находящимся в соответствующей позиции прямой перестановки. Рассмотрим пример работы такого кроссовера (см. рис.2)

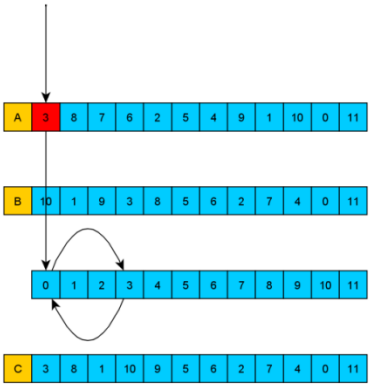

Рисунок 2. Пример работы кроссовера

Оператор кроссовера скрещивает две родительские особи A и B и получает потомка C. На первом шаге происходит сравнение стоимости назначения элементов, находящихся на нулевой позиции перестановки. В данном примере оказывается, что данное назначение наиболее выгодно в родителе A. Поэтому необходимо поставить на нулевую позицию потомка элемент 3. Это происходит путем обмена позиции прямой перестановки, в которой находится элемент 3 и текущей позиции (в данном случае, 0). Аналогичные действия происходят поочередно с каждой позицией родительских особей. Очевидно, что каждое следующее действие может

изменить результаты предыдущих (в нашем примере можно заметить, что на третьей позиции потомка находится элемент 10, а не 0, как это было сделано на первом шаге алгоритма). В результате получаем перестановку, кодирующую потомка C.

Сложность описанного кроссовера составляет $O(n^2)$. Вероятность кроссовера в разрабатываемом алгоритме равна 1, что означает, что каждая брачная пара образует потомка по предложенной схеме.

2.3.3. Мутация

Процедура мутации необходима для внесения разнообразия в популяцию. Мутации подвергается небольшая часть потомков, полученных в результате репродукции. Обычно эта часть определяется параметром генетического алгоритма, который называется вероятностью мутации. Аналогично оператору скрещивания оператор мутации должен гарантировать сохранение перестановки.

Процедура мутации в предлагаемом алгоритме состоит из двух этапов. На первом этапе происходит некоторое заранее заданное количество случайных парных обменов в перестановке потомка. На втором этапе мутант подвергается локальной оптимизации, процедура которой описана ниже.

2.3.4. Отбор решений для следующего поколения и условие останова

Отбор особей для следующего поколения осуществляется по их функции приспособленности, определенное количество лучших особей переходит в следующее поколение. Вычислительная сложность функции оценки приспособленности особей составляет $O(n^2)$, поскольку она представляет собой подсчет критерия КЗН. Условием останова предлагаемого алгоритма является ограничение количества поколений, сгенерированных в ходе решения задачи.

2.3.5. Локальная оптимизация

В ходе работы генетического алгоритма применяется локальная оптимизация решений на основе парных обменов элементов перестановки, кодирующей решение. Механизм локальной оптимизации основан на жадном принципе. Для каждого элемента перестановки находится первый элемент, обмен с которым приводит к уменьшению критерия, и производится парный обмен. Такая процедура происходит некоторое, заранее заданное количество

раз (количество итераций локальной оптимизации). Если в конце очередной итерации не произошло улучшение качества решения, то происходит ряд случайных парных обменов в перестановке, количество которых зависит от номера текущей итерации и от параметра локальной оптимизации, который называется глубиной локальной оптимизации. Такая процедура помогает вырваться из локального оптимума и продолжить поиск лучшего решения.

Сложность описанного алгоритма локальной оптимизации составляет $O(n^4)$.

2.3.6. Схема предлагаемого алгоритма

На рисунке 3 представлен псевдокод предлагаемого генетического алгоритма для решения КЗН.

```
1.  solution GA()
2.  {
3.  permutation[] p = generate();
4.  while (!ga_stop())
5.   {
6.    evaluate(p);
7.    breeding();
8.    crossover();
9.    mutation();
10.   select();
11. }
12. x = record(p);
13. return x;
14. }
```

Рисунок 3. Структура генетического алгоритма

Контракт GA (строка 1) заключается в том, что функция поиска как результат возвращает найденное решение. Решение реализуется сущностью solution – по сути программная конструкция, которая непосредственно соответствует решению задачи. В процессе работы GA (строка 6) контролирует лучшие решения с точки зрения функции приспособленности. GA генерирует начальное поколение p (строка 3), при этом размер поколения неизменен в процессе поиска и является параметром работы GA. При этом каждый элемент поколения описывается перестановкой определенной длины, которая кодирует решение задачи.

Цикл работы GA (строка 4) не прекращается до тех пор, пока не наступят условия останова. Нами в основу функции ga_stop() положена тривиальная

логика, ограничивающая число итераций алгоритма. Цикл (строки 4 – 11) осуществляет традиционный генетический поиск. Процедура evaluate(p) (строка 6) оценивает функцию приспособленности особей. Процедура breeding() (строка 7) выполняет отбор родительских пар. В наших экспериментах мы использовали стратегию панмиксии – право сформировать родительскую пару имеют все особи популяции с равной вероятностью. Процедура crossover() (строка 8) рекомбинацией родительских строк получает две новые строки потомков. Процедура mutation() (строка 9) генерирует случайные изменения в строках потомков. Все новые решения оцениваются, и в процедуре select() (строка 10) происходит замена худших решений из текущего поколения на новые решения, полученные в рамках текущей итерации.

Лучшее найденное решение возвращается как результат работы алгоритма (строка 13).

Итоговая вычислительная сложность предлагаемого генетического алгоритма составляет $O(n^4)$.

2.4. Алгоритм восстановления решения исходной задачи

На первом этапе алгоритма в результате декомпозиции вершины графа параллельной программы были объединены в подграфы, которые были распределены по вычислителям на втором этапе алгоритма. Результатом второго этапа алгоритма стала перестановка длины k (количество вычислителей), описывающая распределение подграфов по вычислителям. Для получения решения исходной задачи, необходимо восстановить первоначальную нумерацию вершин графа параллельной программы и назначить эти вершины на вычислители, на которые были назначены соответствующие подграфы. Таким образом, получим вектор назначения длины *n* (количество частей параллельной программы). Рассмотрим данную процедуру на примере графа параллельной программы, состоящего из 9 вершин, и вычислительной системы, состоящей из 3 вычислителей. На рисунке 4 показано распределение вершин графа по подграфам.

Номера подграфов	Номера вершин		
1	2	3	7
2	4	5	9
3	1	6	8

Рисунок 4. Распределение вершин графа по подграфам

На рисунке 5 приведена перестановка, полученная в результате второго этапа алгоритма и вектор, полученный в результате восстановления решения.

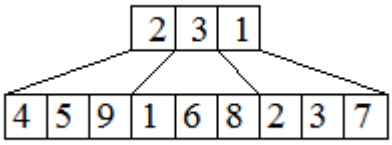

Рисунок 5. Восстановление нумерации

В результате восстановления получаем распределение частей параллельной программы по вычислителям. Однако, в общем случае, данное условие может не удовлетворять балансным ограничениям задачи, что приводит к необходимости применять локальную оптимизацию, которая перераспределит части параллельной программы с целью улучшения сбалансированности решения.

2.5. Локальная оптимизация решения

Целью локальной оптимизации является улучшение качества решения с точки зрения критерия и балансных ограничений. Предлагаемый метод локальной оптимизации основан на подходе, предложенным C.Fiduccia и R.Mattheyses в [7]. Основная идея метода заключается в переносе вершин графа параллельной программы, попавших в сечение при его декомпозиции, из одного подграфа в другой, если это приводит к улучшению качества решения. Поскольку, как говорилось выше, решение, полученное после второго этапа многоуровневого алгоритма, в общем случае может не удовлетворять балансным ограничениям, то во время локальной оптимизации необходимо уделять внимание как значению критерия, так и балансным ограничениям. Параметром алгоритма локальной оптимизации является допустимое отклонение от баланса. Пока оптимизируемое решение превышает это значение, алгоритм работает на улучшение балансного критерия, когда решение удовлетворяет балансному ограничению, алгоритм работает на улучшение значения коммуникационного критерия.

На рисунке 6 представлен псевдокод общей схемы предлагаемого алгоритма локальной оптимизации.

```
1.  Loc(graph G, matrix M, solution x)
2.  {
3.      solution rec = x;
4.      for(int j = 0; j <n1; j++)
5.      {
6.          for(int i = 0; i < n2; i++)
7.          {
8.              if(balance(x) < maxbalance)
9.                  LocCB(G, M, x, rec);
10.             else
11.                 LocBC(G, M, x, rec);
12.         }
13.         rec.CopyTo(x);
14.     }
15.}
```

Рисунок 6. Общая схема алгоритма локальной оптимизации

Алгоритм принимает на вход граф и матрицу, описывающие исходные данные задачи, и решение, полученное в результате первых двух стадий многоуровневого алгоритма (строка 1). Алгоритм состоит из двух вложенных циклов (строки 4-14). Во внешнем цикле происходит переход в локальный оптимум (строка 13), найденный во внутреннем цикле (строки 6-12). Во внутреннем цикле в зависимости от дисбаланса текущего решения происходит его оптимизация с целью уменьшения критерия (строки 8-9) или дисбаланса (строки 10-11).

Рассмотрим схемы этих алгоритмов. На рисунке 7представлена схема алгоритма, уменьшающего коммуникационный критерий.

```
1.  LocCB(graph G, matrix M, solution x, solution rec)
2.  {
3.     List cutNodes = getCutNodex(G, x);
4.     List Move = getBestMoveCut(cutNodes, G, M, x);
5.     for(int i = 0; i < depth; i++)
6.     {
7.        doBestMove(Move, x);
8.        if((Cut(x) < recCut) || ((Cut(x) == recCut)&&(balance(x) < recBal)))
9.        {
10.          recCut = Cut(x);
11.          recBal = balance(x);
12.          x.CopyTo(rec);
13.       }
14.       Move.DeleteBestMove();
15.    }
16. }
```

Рисунок 7. Схема алгоритма, уменьшающего коммуникационный критерий

Алгоритм принимает на вход граф и матрицу, описывающие исходные данные задачи, оптимизируемое решение, а так же рекорд, полученный в ходе работы основного алгоритма локальной оптимизации (строка 1). Предлагаемый алгоритм оперирует перемещением вершин графа параллельной задачи, попавших в сечение, в соседние домены. Поэтому необходимо найти все вершины, попавшие в сечение (строка 3) и найти наиболее выгодное перемещение с точки зрения критерия в соседний домен для каждой вершины (строка 4). В алгоритме задан параметр depth, который определяет количество вершин, которые будут перемешены. В цикле (строки 5-15) последовательно осуществляются наилучшие перемещения вершин, и, если появляется решение лучше рекордного с точки зрения критерия или баланса, то рекорд обновляется (строки 8-13).

На рисунке 8 представлена схема алгоритма, улучшающего балансный критерий.

```
1.  LocBC(graph G, matrix M, solution x, solution rec)
2.  {
3.    List cutNodes = getCutNodex(G, x);
4.    List Move = getBestMoveBal(cutNodes, G, M, x);
5.    for(int i = 0; i < depth; i++)
6.    {
7.      doBestMove(Move, x);
8.      if((balance(x) < recBal) || ((balance(x) == recBal)&&(Cul(x) < recCut)))
9.      {
10.       recCut = Cut(x);
11.       recBal = balance(x);
12.       x.CopyTo(rec);
13.     }
14.     Move.DeleteBestMove();
15.   }
16. }
```

Рисунок 8. Схема алгоритма, улучшающего балансный критерий

Алгоритм принимает на вход граф и матрицу, описывающие исходные данные задачи, оптимизируемое решение, а также рекорд, полученный в ходе работы основного алгоритма локальной оптимизации (строка 1). Предлагаемый алгоритм оперирует перемещением вершин графа параллельной задачи, попавших в сечение, в соседние домены. Поэтому необходимо найти все вершины, попавшие в сечение (строка 3) и найти наиболее выгодное перемещение с точки зрения сбалансированности в соседний домен для каждой вершины (строка 4). В алгоритме задан параметр depth, который определяет количество вершин, которые будут перемешены. В цикле (строки 5-15) последовательно осуществляются наилучшие перемещения вершин, и, если появляется решение лучше рекордного с точки зрения критерия или баланса, то рекорд обновляется (строки 8-13).

Вычислительная сложность описанного алгоритма локальной оптимизации составляет $O(n^3)$.

Литература:

1. Garey M. R., Johnson D. S., Stockmeyer L. Some Simplified NP-complete Problems. Proceedings of the Sixth Annual ACM Symposium on Theory of Computing, 1974, p. 47-63
2. E. Çela. The quadratic assignment problem: Theory and Algorithms. Springer, 31 дек. 1997 г. - Всего страниц: 304
3. Писсанецки С. П34 Технология разреженных матриц: Пер. с англ.—М.: Мир, 1988. —410 с, ил.
4. Старостин Н.В., Филимонов А.В. Разработка и реализация параллельного многоуровневого алгоритма равномерного разбиения нераспределенного графа. Материалы 13-й Всерос. конф. Н.Новгород, 14-16 ноября 2013г. Н.Новгород: Издательство ННГУ, 2013. с.243-247
5. Fiduccia, C., Mattheyses, R.: A linear-time heuristic for improving network partitions. Technical Report 82CRD130, General Electric Co., Corporate Research and Development Center, Schenectady, NY, 1982
6. Батищев Д.И., Костюков В.Е., Неймарк Е.А., Старостин Н.В. Решение дискретных задач с помощью эволюционно-генетических алгоритмов: Учебное пособие. Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2011. 199 с.
7. Fiduccia, C., Mattheyses, R.: A linear-time heuristic for improving network partitions. Technical Report 82CRD130, General Electric Co., Corporate Research and Development Center, Schenectady, NY, 1982

Николай Владимирович **Старостин**

Маргарита Александровна **Быкова**

МНОГОУРОВНЕВЫЙ АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ АРХИТЕКТУРНО-ЗАВИСИМОЙ ДЕКОМПОЗИЦИИ

Учебно-методическое пособие

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

603950, Нижний Новгород, пр. Гагарина, 23.