УДК 519.687.6

МНОГОУРОВНЕВЫЕ АЛГОРИТМЫ ДЕКОМПОЗИЦИИ ГРАФА ДАННЫХ ДЛЯ ПАРАЛЛЕЛЬНЫХ ВЫЧИСЛЕНИЙ НА ГЕТЕРОГЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЕ

Н. В. Старостин, М. А. Панкратова
(ННГУ им. Н. И. Лобачевского, г. Нижний Новгород)

Рассматривается актуальная задача архитектурно-зависимой декомпозиции, позволяющая эффективно планировать выполнение параллельной задачи на многопроцессорной вычислительной системе. Приводится математическая постановка общей задачи архитектурно-зависимой декомпозиции, рассматриваются ее частные случаи, предлагаются два многоуровневых алгоритма решения. Проведено сравнение результатов выполнения алгоритмов на тестовых примерах с результатами известных программных продуктов.

Ключевые слова: архитектурно-зависимая декомпозиция, отображение, параллельная программа, суперкомпьютер, многоуровневый алгоритм.

Введение

Современные вычислительные системы — это ресурс, на котором одновременно могут выполняться несколько заданий. Для его эффективного использования требуется решать задачи, связанные с планированием заданий. Планированием и запуском параллельных задач занимается специальная служба операционной системы (планировщик). Проблему планирования разделяют на два класса — планирование независимых задач и планирование взаимодействующих задач. Планирование независимых задач заключается в сбалансированном распределении задач по вычислителям, планирование взаимодействующих задач — в распределении задач с учетом их информационных обменов, чтобы минимизировать стоимость межпроцессорных коммуникаций. Очевидно, что время выполнения программы будет зависеть от физической топологии выделенного участка вычислительной системы и от распределения параллельных подзадач по вычислителям, поскольку время передачи данных в каждом случае будет разным. Возникает проблема оптимального отображения параллельных подзадач на физическую топологию вычислительной системы [1]. Кроме того, на практике часто возникают ситуации, когда размер подзадач параллельной программы много превышает размер доступной для использования вычислительной системы. В таком случае необходима предварительная декомпозиция параллельной программы на необходимое число частей с целью сбалансированной загрузки вычислителей и минимизации внешних связей. Задачи, связанные с декомпозицией и распределением параллельной программы по узлам вычислительной системы, будем называть задачами архитектурно-зависимой декомпозиции (АЗД).

В работе приводится математическая постановка общей задачи АЗД, рассматриваются ее частные случаи, предлагаются два многоуровневых алгоритма решения поставленной задачи. Первый алгоритм основан на рекурсивной бисекции входных данных задачи, второй — на последовательной релаксации исходных данных и сведении задачи к задаче декомпозиции графа и квадратичной задаче о назначениях. Предложенные алгоритмы были программно реализованы, проведено сравнение результатов их выполнения на тестовых примерах с результатами известных программных продуктов.

Постановка задачи

Исходными данными задачи являются граф данных параллельной программы и вычисли-

тельная система (участок вычислительной системы), выделенная для ее выполнения.

Параллельные программы, на которые ориентирована работа, возникают в области численного моделирования физических объектов и процессов. Для моделирования используются дискретные аппроксимационные сетки, каждый узел которых представляет участок объекта или пространства и хранит в себе его физические характеристики. Связи между узлами сетки определяют влияние одного участка объекта на другой. Рассматриваемые расчетные аппроксимационные сетки носят разреженный неструктурированный характер (порядок количества связей между узлами сетки равен порядку количества узлов, каждый узел сетки может быть связан с любым другим узлом) и характеризуются большим числом узлов (10^6—10^9). В зависимости от количества физических характеристик, которые необходимо рассчитать, узлы сетки характеризуются разным объемом вычислений. Количество вычислителей системы обычно много меньше количества узлов расчетной сетки, что не дает возможности рассчитывать каждый узел сетки в параллельном режиме. Таким образом, необходимо произвести сбалансированную декомпозицию данных расчетной сетки на заданное количество вычислителей и перераспределить полученные части по вычислителям с целью минимизации стоимости внешних связей.

Формально структуру параллельной программы можно описать посредством неориентированного взвешенного графа $G'\left(V', E', w', u'\right)$, где $V' = \{v'_1, \ldots, v'_n\}$ — множество вершин, моделирующих параллельные части программы; $E' \subseteq V'^2$ — множество ребер, моделирующих коммуникационные связи между частями программы; веса вершин $w' : V' \to N$ (N — множество натуральных чисел) моделируют вычислительные издержки на выполнение частей параллельной программы; веса ребер $u' : E' \to N$ моделируют объем коммуникаций между частями параллельной программы.

Существенные с точки зрения рассматриваемой проблемы аспекты вычислительной системы удобно описывать посредством взвешенного гиперграфа $G''\left(V'', E'', w'', u''\right)$, где $V'' = \{v''_1, \ldots, v''_k\}$ — множество вершин, моделирующих вычислители системы; $E'' \subseteq 2^{V''}$ — множество гиперребер, моделирующих коммуникационные связи между вычислителями*; веса вершин $w'' : V'' \to N$ моделируют производительность вычислителей; веса гиперребер $u'' : E'' \to N$ определяют характеристики коммуникационных сред (способ передачи информации — сообщениями или пакетами, время подготовки информации к отправке, скорость передачи единицы информации, время передачи служебной информации и т. п.).

Гиперграфовая модель сопоставляет компонентам и характеристикам вычислительной системы математические сущности, что удобно при формировании исходных данных. Однако для постановки задачи эта модель не очень удобна, так как не содержит в явном виде данных о затратах, которые возникают при произвольных межпроцессорных коммуникациях. Формально можно ввести функции времени реализации транзакции $T_{ij}(m) = T\left(G'', i, j, m\right)$ в топологии G'', где i и j — номера вычислителей $i, j = \overline{1,k}$; m — объем пересылаемых данных. Для описания времени реализации межпроцессорной пересылки данных будем использовать матрицу $S = \{s_{ij}\}_{k\times k}$, где коэффициенты $s_{ij} \in N$, $i, j = \overline{1,k}$, соответствуют коэффициентам линейной аппроксимации функций $T_{ij}(m)$. В этом случае время передачи информации объемом m между вычислителями i и j можно оценивать по формуле $t(i, j, m) = s_{ij}m$.

В качестве решения задачи выберем вектор $x = \{x_1, \ldots, x_n\}$, $x_i \in \{1, \ldots, k\}$, $i = \overline{1,n}$, где i-й элемент соответствует части параллельной программы, а x_i — номер вычислителя, на котором она будет выполняться. Определим ограничения математической модели.

Рассчитаем вычислительную нагрузку на вычислитель i по формуле

$$W_i(x) = \sum_{x_j=i} w'\left(v'_j\right), \quad i = \overline{1,k}.$$

Рассчитаем идеальную вычислительную нагрузку на вычислитель i по формуле

$$\widetilde{W}_i = \frac{\sum\limits_{j=1}^{n} w'\left(v'_j\right)}{\sum\limits_{j=1}^{k} w''\left(v''_j\right)} w''\left(v''_i\right), \quad i = \overline{1,k}.$$

*Если множество вычислителей соединены коммуникатором или шиной и могут обмениваться друг с другом информацией с одинаковой скоростью, то множество вершин гиперграфа, соответствующих этим вычислителям, образуют гиперребро.

Введем параметр $\varepsilon \in [0, 1]$, который задает максимальное отклонение от баланса. Тогда можно записать ограничение на максимальное отклонение нагрузки вычислителей от идеального значения:

$$\max_{i=\overline{1,k}} \left(\left| \frac{W_i(x)}{\widetilde{W}_i} - 1 \right| \right) < \varepsilon. \quad (1)$$

Можно ограничивать среднее отклонение нагрузки от идеального значения. Тогда балансное ограничение выглядит следующим образом:

$$\frac{1}{k} \sum_{i=1}^{k} \left(\left| \frac{W_i(x)}{\widetilde{W}_i} - 1 \right| \right) < \varepsilon. \quad (2)$$

Проблема моделирования коммуникационных обменов в программе, выполняющейся в параллельной вычислительной системе, связана с тем, что 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 [2].

Определим функцию

$$\beta\left(x, v_i', v_j'\right) = \begin{cases} u'\left(v_i', v_j'\right) s_{x_i x_j}, & \left(v_i', v_j'\right) \in E'; \\ 0, & \left(v_i', v_j'\right) \notin E', \end{cases}$$

для оценки времени коммуникационного обмена между парой параллельных частей программы. Нижняя оценка предполагает, что все обмены происходят одновременно и друг с другом не конфликтуют. Тогда имеет место минимаксный критерий

$$F_1(x) = \max_{\left(v_i', v_j'\right) \in E'} \beta\left(x, v_i', v_j'\right) \to \min. \quad (3)$$

Верхняя оценка предполагает, что все обмены конфликтуют друг с другом и происходят последовательно. При этом имеет место аддитивный критерий

$$F_2(x) = \sum_{\left(v_i', v_j'\right) \in E'} \beta\left(x, v_i', v_j'\right) \to \min. \quad (4)$$

Задачи, в которых присутствуют ограничения вида (1) и/или (2) и критерии (3) и/или (4), будем называть задачами АЗД.

С практической точки зрения интересны два частных случая задачи АЗД.

Первый случай — когда имеет место гомогенная вычислительная система с равномощными процессорами и одинаковыми затратами на передачу данных между любой парой вычислителей:

$$n > k;\ w'' \equiv 1;\ s_{ij} = 1 \quad \forall i,\ j = \overline{1,k},\ i \neq j. \quad (5)$$

При этом задачу АЗД (1), (5), (4) можно сформулировать в терминах задачи сбалансированного k-разбиения графа (СРГ) [3]: требуется распределить параллельные части программы по k равновзвешенным подграфам (соответствуют процессорам) так, чтобы связи между подграфами были минимальными.

Другой частный случай имеет место, когда число частей параллельной программы и число вычислителей одинаково, при этом процессоры равномощны, но затраты на передачу данных различаются:

$$n = k;\ w'' \equiv 1;\ s_{ij} > 0 \quad \forall i,\ j = \overline{1,k},\ i \neq j. \quad (6)$$

Тогда требования (1) сбалансированной загрузки вычислителей вырождаются в требование взаимнооднозначного соответствия между частями параллельной программы и вычислителями и задача АЗД (1), (6), (4) формулируется в терминах квадратичной задачи о назначениях (КЗН) [4].

Рассмотренные задачи АЗД и обозначенные частные случаи СРГ и КЗН в общем случае являются NP-трудными [4, 5].

Многоуровневые схемы решения задач АЗД

Прикладные задачи АЗД характеризуются такими большими размерами исходных данных, что вычислительные издержки для целого ряда приближенных алгоритмов (не говоря уже о точных) ограничивают возможности их практического применения. Предлагаются два алгоритма, основанные на концепции многоуровневости, — от исходной задачи строится серия или последовательность более простых задач, решение которых позволяет синтезировать или восстановить решение исходной задачи.

Первый алгоритм использует идею рекурсивной бисекции, которая состоит из следующих шагов:

- разбиение графа данных параллельной программы на два подграфа с минимальными внешними связями (задача сбалансированного биразбиения графа);
- разбиение матричной модели вычислительной системы на два блока (подматрицы) минимальными межблочными связями (задача сбалансированного биразбиения графа матрицы);
- назначение полученных частей графа на блоки матриц (квадратичная задача о назначениях размерности 2).

В результате происходит генерация двух задач АЗД с вдвое меньшими размерами. К полученным задачам можно рекурсивно применить описанную процедуру и еще уменьшить размеры исходных данных.

На рис. 1 показана общая схема рекурсивной бисекции. На первом уровне исходная задача АЗД разбивается на две задачи: АЗД1 и АЗД2; на втором уровне каждая из задач АЗД1 и АЗД2, в свою очередь, разбивается на две подзадачи (АЗД11, АЗД12 и АЗД21, АЗД22) и т. д. до тех пор, пока не будут получены задачи АЗД приемлемого порядка. В пределе можно дробить задачи АЗД до получения тривиального случая, когда размер матрицы вычислительной системы станет равным единице и произойдет непосредственное назначение группы вершин графа данных параллельной программы на вычислительный узел.

Бисекцию графа предлагается осуществлять, используя итерационный многоуровневый алгоритм [6]. Идея алгоритма заключается в многоуровневой редукции (уменьшении размеров) задачи, поиске решения уменьшенной задачи и последовательном его восстановлении и проекции на исходные данные с локальной оптимизацией. Итерационность заключается в том, что данная процедура происходит не один раз и каждая следующая итерация использует результат, полученный на предыдущей итерации.

Бисекцию матрицы предлагается осуществлять с помощью спектрального алгоритма [7]. Идея алгоритма заключается в построении матрицы Лапласа по элементам исходной матрицы, расчете спектрального вектора, который упорядочивает строки и столбцы матрицы по мере их удаленности друг от друга.

На рис. 2 приведен псевдокод алгоритма, реализующего схему рекурсивной бисекции. Алгоритм принимает на вход граф, моделирую-

Рис. 1. Схема рекурсивной бисекции

```
1. Algorithm1(graph G, matrix T, out int[] X)
2. {
3.    if(size(T)==1)
4.       map(G, T, out X);
5.    else
6.    {
7.       bisect(G) -> graph G1, graph G2;
8.       spectrbisect(T, G1, G2) -> matrix T1, matrix T2;
9.       Algorithm1(G1, T1, X);
10.      Algorithm1(G2, T2, X);
11.   }
12. }
```

Рис. 2. Алгоритм рекурсивной бисекции

щий параллельную программу, матрицу, моделирующую вычислительную систему, и сохраняет полученное решение в массиве (строка 1). В основной части алгоритма происходит бисекция графа с использованием многоуровневого итерационного алгоритма (строка 7), бисекция матрицы спектральным методом [7] пропорционально произведенной бисекции графа (строка 8), и алгоритм рекурсивно повторяется на полученных частях графа и матрицы (строки 9, 10). В случае, если размер вычислительной системы оказался равным 1, бисекция не выполняется, а происходит назначение всех вершин соответствующего подграфа параллельной программы на данный вычислительный узел (строки 3, 4).

Второй предлагаемый алгоритм использует концепцию последовательного сведения исходной задачи АЗД к КЗН приемлемого размера. Основная идея алгоритма состоит в том, что на одном вычислительном узле с большей вероятностью будут выполняться те части параллельной программы, которые интенсивно обмениваются информацией между собой. Поэтому предлагается произвести декомпозицию графа параллельной программы на количество частей, равное количеству вычислителей, с целью минимизации внешних связей между подграфами разбиения. Такая предварительная декомпозиция заранее гарантирует, что самые интенсивные обмены данными будут происходить в рамках одного вычислителя и не будут занимать коммуникационную среду. После декомпозиции графа параллельной программы необходимо назначить полученные подграфы на узлы вычислительной системы, что приводит к квадратичной задаче о назначениях. На рис. 3 показана общая схема предлагаемого алгоритма.

Переход от исходной задачи к задаче декомпозиции (см. рис. 3) заключается в релаксации исходных данных; все вычислители считаются равномощными и стоимость передачи данных между каждой парой вычислителей считается одинаковой. Решение задачи сбалансированной декомпозиции графа на k подграфов осуществляется с использованием многоуровневого итерационного алгоритма [6].

По полученному разбиению строится супер-граф, где каждая супервершина ассоциируется с подграфом декомпозиции, а ребро между двумя супервершинами существует в том случае, если хотя бы две вершины соответствующих подграфов были соединены ребром в исходном графе; при этом вес ребра определяется как сумма весов ребер, связывающих вершины из соответствующих подграфов. Размер построенного супер-графа совпадает с количеством вычислителей системы. На этапе перехода к КЗН (см. рис. 3) используется исходная информация о стоимости передачи данных между вычислителями, мощности вычислителей по-прежнему считаются равными. Решение КЗН предлагается осуществлять по генетическому алгоритму [8].

На рис. 4 приведен псевдокод алгоритма, реализующего описанный подход к решению поставленной задачи. Алгоритм принимает на вход граф, моделирующий параллельную программу, матрицу, моделирующую вычислительную систему, и сохраняет полученное решение в массиве (строка 1). В первой части алгоритма происходит декомпозиция графа на количество частей, соответствующее размеру матрицы (строка 3). Декомпозиция осуществляется с помощью многоуровневого итерационного алгоритма. Затем строится супер-граф, вершины которого соответствуют частям декомпозированной сетки, а ребро между вершинами существует в том случае, если в графе расчетной сетки между соответствующими частями существовало хотя бы одно ребро (строка 4). Во второй части алгоритма (строка 5) происходит перераспределение частей декомпозированного графа по узлам вычислительной системы с использованием генетического алгоритма.

Решение КЗН технически просто сформулировать в терминах задачи АЗД, что предполагает восстановление нумерации вершин исходного

Рис. 3. Схема многоуровневого алгоритма

```
1. Algorithm2(graph G, matrix T, out int[] X)
2. {
3.     partition p = decomp(G, size(T));
4.     graph g = coars(G, p);
5.     GA(g, T, out X);
6. }
```

Рис. 4. Двухуровневый алгоритм

графа. Главная проблема в том, что результат может в общем случае не удовлетворять ограничениям баланса (1) и (2). Это происходит из-за того, что на этапе декомпозиции задачи АЗД не учитывались мощности вычислителей. Для балансировки загрузки вычислителей предлагается процедура локальной оптимизации, основанная на аналоге алгоритма из работы [9]. Суть алгоритма состоит в итерационном процессе, который на каждой итерации осуществляет серию балансирующих переносов вершин между подграфами.

Вычислительный эксперимент

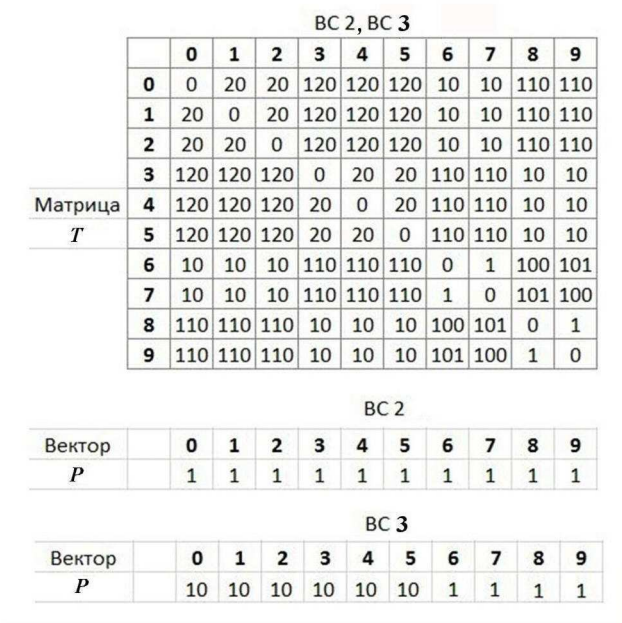
Для вычислительного эксперимента были выбраны три примера вычислительных сетей (ВС) и набор сеток в качестве графов параллельных программ. Примеры сеток были взяты из библиотек университетов Гринвича [10] и Флориды [11]. На рис. 5, 6 приведены примеры используемых ВС, где показаны матрицы T относительных издержек на передачу данных между вычислителями и векторы P относительных производительностей вычислителей.

ВС 1

Матрица T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	2	2	2	2	2	2	2	4	4	4	4	4	4	4	4
1	2	0	2	2	2	2	2	2	4	4	4	4	4	4	4	4
2	2	2	0	2	2	2	2	2	4	4	4	4	4	4	4	4
3	2	2	2	0	2	2	2	2	4	4	4	4	4	4	4	4
4	2	2	2	2	0	2	2	2	4	4	4	4	4	4	4	4
5	2	2	2	2	2	0	2	2	4	4	4	4	4	4	4	4
6	2	2	2	2	2	2	0	2	4	4	4	4	4	4	4	4
7	2	2	2	2	2	2	2	0	4	4	4	4	4	4	4	4
8	4	4	4	4	4	4	4	4	0	2	2	2	2	2	2	2
9	4	4	4	4	4	4	4	4	2	0	2	2	2	2	2	2
10	4	4	4	4	4	4	4	4	2	2	0	2	2	2	2	2
11	4	4	4	4	4	4	4	4	2	2	2	0	2	2	2	2
12	4	4	4	4	4	4	4	4	2	2	2	2	0	2	2	2
13	4	4	4	4	4	4	4	4	2	2	2	2	2	0	2	2
14	4	4	4	4	4	4	4	4	2	2	2	2	2	2	0	2
15	4	4	4	4	4	4	4	4	2	2	2	2	2	2	2	0

Вектор P	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Рис. 5. Пример ВС 1



ВС 2, ВС 3

Матрица T	0	1	2	3	4	5	6	7	8	9
0	0	20	20	120	120	120	10	10	110	110
1	20	0	20	120	120	120	10	10	110	110
2	20	20	0	120	120	120	10	10	110	110
3	120	120	120	0	20	20	110	110	10	10
4	120	120	120	20	0	20	110	110	10	10
5	120	120	120	20	20	0	110	110	10	10
6	10	10	10	110	110	110	0	1	100	101
7	10	10	10	110	110	110	1	0	101	100
8	110	110	110	10	10	10	100	101	0	1
9	110	110	110	10	10	10	101	100	1	0

ВС 2

Вектор P	0	1	2	3	4	5	6	7	8	9
	1	1	1	1	1	1	1	1	1	1

ВС 3

Вектор P	0	1	2	3	4	5	6	7	8	9
	10	10	10	10	10	10	1	1	1	1

Рис. 6. Примеры ВС 2 и ВС 3

Пример 1 (см. рис. 5) построен на основе структуры *жирного дерева*; примеры 2 и 3 (см. рис. 6) — на основе симметричной блочной структуры и отличаются между собой производительностью вычислителей.

Целью первой части вычислительного эксперимента было подтверждение гипотезы о положительном влиянии предварительной декомпозиции графа параллельной программы на качество решения. Эксперимент был проведен на известном программном продукте Scotch [2]. Сравнивались результаты, полученные данной программой без предварительной декомпозиции графа параллельной программы и при выполнении предварительной декомпозиции.

Полученные результаты представлены в табл. 1. Видно, что для большей части тестовых примеров удалось уменьшить значение критерия (4) с помощью предварительной декомпозиции. Следует отметить, что при этом несколько увеличивается разбалансированность итогового разбиения задачи, однако значения разбалансированности находятся в "разумных" пределах с точки зрения практического применения полученного решения.

Вторая часть эксперимента посвящена сравнению результатов, полученных с помощью предложенных алгоритмов, с результатами Scotch. Результаты экспериментов (табл. 2) показали хорошее качество решений по предложенным алгоритмам. На ряде тестовых примеров удалось получить лучшие решения с точки зрения критерия (4) по сравнению с решениями, полученными Scotch. Из анализа табл. 2 также можно сделать вывод об эффективной работе предложенной схемы локальной оптимизации, позволяющей улучшить как значение критерия, так и уменьшить дисбаланс загрузки вычислителей.

Отметим, что целью данной работы была разработка алгоритма решения поставленной задачи, поэтому для вычислительного эксперимента был написан академический неоптимизированный программный код. В связи с этим сравнение времени выполнения тестируемых алгоритмов здесь не производится.

Таблица 1

Влияние предварительной декомпозиции на качество решения

Сетка	Размер	ВС	Scotch			Декомпозиция+Scotch		
			Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса	Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса
gr_30_30	900	1	645	1,33	0,67	602	13,8	7,9
bcsstk29	13 992	1	32 270	0,97	0,77	27 142	5,55	1,95
bcsstk32	44 609	1	55 491	0,97	0,91	43 877	2,8	1,8
bcsstk33	8 738	1	70 401	1,12	0,61	69 265	15,04	5,7
barth5	15 606	1	1 212	0,96	0,58	1 131	4,96	3,08
vibrobox	12 328	1	50 705	1,36	0,6	56 091	4,99	3,76
ef_sphere	16 386	1	2 438	0,99	0,58	2 550	4,99	4
ef_4elt2	11 143	1	1 294	0,64	0,15	1 191	4,8	3,1
fe_rotor	99 617	1	25 236	0,99	0,83	27 800	4,96	2,7
star_100_100_1	10 000	2	15 089	0,8	0,16	16 575	2	1,4
cube.1e4	11 165	2	40 568	0,49	0,15	50 501	4,34	1,39

Заключение

Рассмотрена актуальная задача АЗД. Предложена общая математическая постановка задачи. Разработаны два алгоритма, позволяющие упрощать задачу путем ее декомпозиции на меньшие задачи, а также сводящие ее к другим задачам. Алгоритмы программно реализованы, проведенный вычислительный эксперимент подтвердил эффективность предложенных схем.

Отметим, что практическая польза от алгоритмов решения задачи АЗД будет гораздо больше при их параллельной реализации. Перспективным направлением развития данного исследования видится разработка способов распараллеливания предложенных алгоритмов. Так, например, при использовании рекурсивной бисекции каждая ветвь может выполняться независимо.

Список литературы

1. *Bokhari S.* On the mapping problem // Computers, IEEE Transactions on. 1981. Vol. 100, No 3. P. 207—214.
2. *Pellegrini F.* Scotch and LibScotch 6.0 User's guide. University of Bordeaux 1 & LaBRI, France, 2012. http://gforge.inria.fr/docman/?group_id=248 &view=listfile&dirid=326.
3. *Батищев Д. И., Старостин Н. В.* k-разбиение графов // Вестник ННГУ. Математическое моделирование и оптимальное управление. Н. Новгород: Изд-во ННГУ, 2000. С. 27—35.
4. *Cela E.* The Quadratic Assignment Problem: Theory and Algorithms. The Netherlands, Dordrecht: Kluwer Academic Publishers, 1998.
5. *Батищев Д. И., Коган Д. И.* Вычислительная сложность экстремальных задач переборного типа. Н. Новгород: Изд-во ННГУ, 1994.
6. *Старостин Н. В., Филимонов А. В.* Разработка и реализация параллельного многоуровневого алгоритма равномерного разбиения нераспределенного графа // Мат. 13-й Всерос. конф. Н. Новгород, 14—16 ноября 2013 г. Н. Новгород: Изд-во ННГУ, 2013. С. 243—247.
7. *Якобовский М. В.* Распределенные системы и сети. Уч. пособие. М.: МГТУ "Станкин", 2000.
8. *Старостин Н. В., Панкратова М. А.* Генетические алгоритмы решения задачи отображения графа // Вестник ННГУ им. Н. И. Лобачевского. 2013. Т. 5(1). С. 204—209.
9. *Fiduccia C. M., Mattheyses R. M.* A linear-time heuristic for improving network partitions // Proc. 19th Conf. on Design Automation. USA, NJ: IEEE Press Piscataway, 1982. P. 175—181.
10. The Graph Partitioning Archive. University of Greenvich. http://staffweb.cms.gre.ac.uk/~c.walshaw/partition/#graphs
11. The University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/

Статья поступила в редакцию 27.05.15.

MULTILEVEL ALGORITHMS OF DECOMPOSING A DATA GRAPH FOR PARALLEL SIMULATIONS ON A HETEROGENOUS COMPUTER SYSTEM / N. V. Starostin, M. A. Pankratova (N. I. Lobachevskii NNSU, Nizhny Novgorod).

The paper considers the urgent problem of the architecture-dependent decomposition that allows effectively planning the execution of a parallel job on a multiprocessor. A mathematical formulation of the general problem of architecture-dependent decomposition is given, its special cases are discussed, and two multilevel computational algorithms are presented. The algorithm execution results using test problems are compared with those obtained with the well-known software products.

Keywords: architecture-dependent decomposition, mapping, parallel code, supercomputer, multilevel algorithm.

Таблица 2

Сравнение предложенных алгоритмов с алгоритмом Scotch

Сетка	Размер	ВС	Алгоритм 1			Алгоритм 2			Алгоритм 2 + балансировка			Scotch		
			Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса	Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса	Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса	Критерий	Максимальное отклонение от баланса	Среднее отклонение от баланса
gr_30_30	900	1	882	70,7	23	600	13,8	7,9	632	5,8	3,5	645	1,33	0,67
bcsstk29	13 992	1	39 043	67,4	15	27 142	5,55	1,95	27 384	4,29	1,95	32 270	0,97	0,77
bcsstk32	44 609	1	65 070	47,5	15,4	43 877	2,8	1,8	43 877	2,8	1,8	55 491	0,97	0,91
bcsstk33	8 738	1	92 670	48,3	26	69 265	15,04	5,7	69 584	10,7	5,3	70 401	1,12	0,61
barth5	15 606	1	1 452	40,4	16,3	1 131	4,96	3,08	1 256	4,37	2,68	1 212	0,96	0,58
vibrobox	12 328	1	63 050	54,6	24,6	52 352	4,99	3,76	52 352	4,99	3,76	50 705	1,36	0,6
ef_sphere	16 386	1	2 959	63,6	23,2	2 354	4,99	4	2 354	4,99	4	2 438	0,99	0,58
ef_4elt2	11 143	1	1 813	45,9	13,8	1 191	4,8	3,1	1 802	2,7	1,8	1 294	0,64	0,15
fe_rotor	99 617	1	37 971	49,5	16,9	24 082	4,96	2,7	33 992	3,18	1,76	25 236	0,99	0,83
star_100_100_1	10 000	2	31 708	42,1	22,3	15 246	2	1,4	15 446	1,6	1,32	15 089	0,8	0,16
star_100_100_1	10 000	3	31 708	636	220	15 246	552,8	236,3	20 857	2,8	1,9	17 336	0,48	0,19
cube.1e4	11 165	2	58 392	38,2	25,2	41 156	4,34	1,39	41 156	4,34	1,39	40 568	0,49	0,15
cube.1e4	11 165	3	58 392	674	245	41 156	567,8	240,6	56 247	7,19	5,85	46 946	0,31	0,13