РОССИЙСКАЯ АКАДЕМИЯ НАУК

СИБИРСКОЕ ОТДЕЛЕНИЕ

А В Т О М Е Т Р И Я

2005, том 41, № 5

ИНФОРМАЦИОННО-ИЗМЕРИТЕЛЬНЫЕ СИСТЕМЫ

УДК 681.3.06 : 681.323

М. С. Тарков

(Новосибирск)

ДЕЦЕНТРАЛИЗОВАННОЕ УПРАВЛЕНИЕ РЕСУРСАМИ И ЗАДАНИЯМИ В ЖИВУЧИХ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

Предложен подход к разработке децентрализованных алгоритмов управления ресурсами и заданиями в живучих распределенных вычислительных системах (ВС), позволяющий организовать согласованное одновременное выполнение множества перестановочных операций на структурах данных, распределенных по машинам ВС. Этот подход сводится к динамическому представлению ВС в виде согласованно и циклически функционирующих непересекающихся подсистем (доменов), в каждой из которых имеется модуль, управляющий процессом оптимизации критерия, заданного на подсистеме. Разбиение ВС на домены изменяется динамически таким образом, что все модули системы получают право управления другими модулями своего домена одинаково часто. Предложенный подход продемонстрирован на примерах построения децентрализованных алгоритмов: 1) оптимизации разбиения ВС на подсистемы для решения набора параллельных задач, 2) вложения структуры параллельной программы в структуру ВС (подсистемы ВС).

Введение. Живучая распределенная вычислительная система (ВС) [1–7] представляет собой множество элементарных машин (ЭМ), связанных сетью, программно управляемой этими машинами. Каждая элементарная машина включает в себя вычислительный модуль (ВМ) и системное устройство. Системное устройство функционирует под управлением ВМ и имеет входные и выходные полюсы, связанные соответственно с выходными и входными полюсами соседних ЭМ. Свойство живучести ВС означает, что отказы и восстановления ее ЭМ приводят только к увеличению и уменьшению времени решения задачи [2]. Управление ресурсами и заданиями в живучих распределенных ВС предусматривает решение следующих задач [1–7]: разбиение ВС на подсистемы; вложение структур параллельных программ в структуры подсистем; статическая и динамическая балансировка (выравнивание) вычислительной нагрузки между вычислительными модулями ВС (или подсистемы ВС); статическая и динамическая маршрутизация (прокладка путей передачи данных), т. е. выравнивание коммуникационной на-

грузки в сети связи ВС; размещение копий программ и данных при организации отказоустойчивых вычислений; реконфигурация подсистем, перераспределение вычислительной и коммуникационной нагрузки с целью восстановления вычислений при отказах и т. п.

Все эти задачи, как правило, рассматриваются как задачи комбинаторной оптимизации [8], решение которых предполагает централизованное выполнение некоторых перестановочных операций на структурах данных, распределенных по элементарным машинам ВС. Централизованный способ решения таких задач предполагает сбор данных в некоторой (центральной) ЭМ, решение оптимизационной задачи в этой ЭМ с последующей рассылкой результатов по всем ЭМ системы (подсистемы). В итоге мы имеем последовательный (и, значит, медленный) способ решения задачи с большими коммуникационными затратами на сбор и рассылку данных. В настоящее время большое развитие получил децентрализованный подход к решению задач управления ресурсами и заданиями в вычислительных системах с распределенной памятью [1, 3], что во многих случаях позволило распараллелить решение этих задач. В то же время остается нерешенной проблема организации согласованного одновременного выполнения множества перестановочных операций в пределах всей ВС без предварительного разбиения ее на подсистемы. В данной работе предлагается новый подход к разработке децентрализованных алгоритмов решения подобных задач, позволяющий решить указанную проблему.

Такой подход не требует предварительного создания каких-либо управляющих структур данных, распределенных по машинам ВС, и может быть реализован на ВС с произвольной структурой, что важно для организации параллельных вычислений, устойчивых к отказам вычислительных модулей и межмодульных соединений. Предполагается, что каждая вершина имеет уникальный идентификатор и на множестве идентификаторов определено отношение полного порядка. Предложенный подход продемонстрирован на примере разработки новых децентрализованных алгоритмов планирования ресурсов и заданий.

Алгоритм согласования пересекающихся подсистем параллельных оптимизационных процессов. Структура вычислительных систем описывается графом $G_s(V_s, E_s)$, где V_s – множество вершин (ЭМ) и $E_s \subseteq V_s \times V_s$ – множество ребер (связей между ЭМ). Под окрестностью $\varepsilon_i(r)$ вершины i понимается множество вершин, удаленных от этой вершины на расстояние, не превышающее радиус r. Пусть радиус r окрестности не зависит от номера вершины i. Связный подграф графа ВС, включающий вершину i и ее окрестность, назовем доменом.

В предлагаемом подходе глобальное преобразование данных на графе ВС сводится к множеству локальных преобразований на доменах графа ВС. Из локальности этих преобразований вытекает возможность их параллельного выполнения, а именно параллельно выполняются локальные преобразования на непересекающихся доменах. При этом возникает задача планирования (согласования, задания порядка выполнения) пересекающихся доменов. Эта задача решается приведенным далее полностью децентрализованным алгоритмом.

При реализации алгоритма согласования доменов (АСД) каждая ЭМ может находиться в одном из следующих состояний: Free, Master, Slave (по отношению к перестановочной операции, выполняемой на домене). На рис. 1 схематически показаны переходы $\{R_1, \ldots, R_4\}$ между этими состояниями.

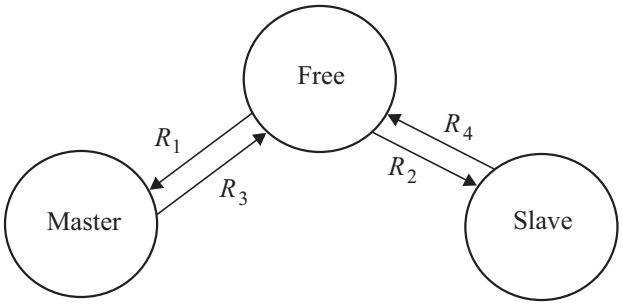

Рис 1. Состояния ЭМ при выполнении алгоритма согласования доменов

Представим правила АСД $\{R_1, \ldots, R_4\}$ с соответствующими комментариями.

$$R_1: \mathrm{State}_i = \mathrm{Free} \wedge \mathrm{Count}_i = 0 \wedge$$
$$\wedge (\forall j \in \varepsilon_i: \mathrm{State}_j = \mathrm{Slave} \vee (\mathrm{State}_j = \mathrm{Free} \wedge$$
$$\wedge (\mathrm{Id}_j > \mathrm{Id}_i \vee (\mathrm{Id}_j < \mathrm{Id}_i \wedge \mathrm{Count}_j > 0)))) \rightarrow$$
$$\rightarrow \mathrm{State}_i := \mathrm{Master}, \mathrm{Master\_Id}_i := \mathrm{Id}_i, \mathrm{Work}_i := \mathrm{true}.$$

Если вершина i свободна (Free), $\mathrm{Count}_i = 0$ и для всех вершин $j \in \varepsilon_i$ выполнено одно из условий:

1) либо вершина является подчиненной (Slave);

2) либо вершина свободна (Free) и ее идентификатор Id_j больше идентификатора Id_i вершины i;

3) либо вершина свободна (Free), ее идентификатор меньше идентификатора вершины i, но $\mathrm{Count}_j > 0$,

то вершина i становится ведущей (Master) и инициирует исполнение алгоритма управления на своем домене $(\mathrm{Work}_i := \mathrm{true})$.

$$R_2: \mathrm{State}_i = \mathrm{Free} \wedge (\exists\, \mathrm{Id}_k = \min_{j \in \varepsilon_i \wedge \mathrm{State}_j = \mathrm{Master}} \mathrm{Id}_j) \rightarrow$$
$$\rightarrow \mathrm{State}_i := \mathrm{Slave}(k), \mathrm{Master\_Id}_i := \mathrm{Id}_k,$$
$$\mathrm{Count}_i := \max(\mathrm{Count}_i - 1, 0), \ \mathrm{Work}_i := \mathrm{true}.$$

Если вершина i свободна (находится в состоянии Free) и в ее окрестности имеются ведущие вершины, то среди них выбирается вершина k с минимальным идентификатором Id_k, для которой вершина i становится подчиненной $(\mathrm{State}_i := \mathrm{Slave}(k))$ и включается в исполнение алгоритма управления. При этом, если $\mathrm{Count}_i > 0$, Count_i уменьшается на единицу. Параметр $\mathrm{Master\_Id}_i$ идентифицирует ведущую вершину k, которой подчиняется вершина i.

$$R_3: \mathrm{State}_i = \mathrm{Master} \wedge \mathrm{Work}_i = \mathrm{false} \rightarrow \mathrm{State}_i := \mathrm{Free}, \mathrm{Count}_i := |\varepsilon_i|.$$

Если вершина i является ведущей и выполнение шага алгоритма управления завершено, то вершина i переходит в состояние Free со счетчиком Count_i, равным мощности $|\varepsilon_i|$ окрестности вершины.

$$R_4:\ \mathrm{State}_i = \mathrm{Slave} \wedge \mathrm{Work}_i = \mathrm{false} \rightarrow \mathrm{State}_i := \mathrm{Free}.$$

Если вершина *i* является подчиненной и выполнение шага алгоритма управления завершено, то вершина *i* переходит в состояние Free (становится свободной).

Алгоритм согласования доменов реализован как циклически выполняемая последовательность правил. Каждое правило включает в себя предикат и действие – последовательность операций, которая выполняется только в случае истинности этого предиката. Если же предикат ложен, то происходит переход к анализу предиката следующего правила. Истинность предиката в каждой вершине графа системы определяется ее собственным состоянием и состояниями вершин в ее окрестности.

В начальный момент все вершины графа системы находятся в состоянии Free и флаг Work в каждой вершине имеет значение false. Вершина управляет выполнением шага алгоритма управления в ее окрестности, когда она находится в состоянии Master, а все остальные вершины в ее окрестности – в состоянии Slave. Переход вершины в (рабочее) состояние Master (правило R_1) или Slave (правило R_2) сопровождается установкой в ней флага Work = true. По окончании шага алгоритма управления все вершины, участвующие в нем, переходят в состояние Free, а флаг Work снова получает значение false.

При переходе вершины *i* из состояния Master в состояние Free (правило R_3) счетчику Count_i присваивается значение числа вершин $|\varepsilon_i|$ в окрестности вершины *i*. Смысл введения счетчика Count_i заключается в том, что вершина *i* вновь сможет стать ведущей только после того, как она $|\varepsilon_i|$ раз выполнит шаг алгоритма планирования, каждый раз в подчинении у одной из $|\varepsilon_i|$ вершин окрестности. Из состояния Slave вершина *i* переходит в состояние Free в соответствии с правилом R_4.

О п р е д е л е н и е. Будем говорить об однократном срабатывании АСД в вершине *i*, если при одноразовом строго последовательном вычислении предикатов в вершине *i* хотя бы один предикат является истинным.

Утверждение 1. Функционирование системы параллельных процессов, реализующей АСД, можно представить как последовательность шагов, на каждом из которых не менее одной вершины срабатывает в состоянии Master, т. е. АСД не имеет тупиковых состояний.

Д о к а з а т е л ь с т в о.

1. Рассмотрим упорядоченную по возрастанию последовательность $\{\mathrm{Id}_0, \mathrm{Id}_1, \ldots, \mathrm{Id}_{|V_s|-1}\}$ идентификаторов вершин графа ВС. Поскольку все идентификаторы различны, то среди них есть минимальный Id_0, а поскольку все вершины изначально находятся в состоянии Free, то вершина с Id_0 согласно правилу R_1 перейдет в состояние Master и все вершины в ее окрестности, в силу независимости радиуса окрестности от значения идентификатора вершины, согласно правилу R_2 перейдут в состояние Slave. По окончании выполнения шага алгоритма управления ($\mathrm{Work}_i = \mathrm{false}$) вершина Id_0 из состояния Master перейдет согласно правилу R_3 в состояние Free с установкой счетчика $\mathrm{Count}_0 = |\varepsilon_0|$ (при этом вершины ее окрестности ε_0 согласно правилу R_4 также перейдут в состояние Free), и согласно правилу R_2 она вновь перейдет в состояние Master после того, как все вершины ее окрестности побывают в состоянии Master и соответственно счетчик Count_0 уменьшится до нуля.

2. Пусть вершины $\mathrm{Id}_0, \ldots, \mathrm{Id}_k$, $k \in \{0,1,\ldots,|V_s|-1\}$, сработали в состоянии Master. Тогда согласно правилу R_1 вершина Id_{k+1} сможет войти в это же состояние (а вершины ее окрестности согласно правилу R_2 перейдут в состояние Slave), поскольку все вершины Id_j множества $\{\mathrm{Id}_0, \ldots, \mathrm{Id}_k\} \cap \varepsilon_{k+1}$ уже сработали в состоянии Master и согласно правилу R_3 находятся в состоянии Free с $\mathrm{Count}_j = |\varepsilon_j|$. Они смогут снова войти в состояние Master не ранее момента срабатывания вершины Id_{k+1} в этом состоянии. В силу произвольности k все вершины $\mathrm{Id}_0, \mathrm{Id}_1, \ldots, \mathrm{Id}_{|V_s|-1}$ сработают в состоянии Master. Это означает, что в таком состоянии могут сработать все вершины окрестности ε_0 вершины Id_0, и после этого вершина Id_0 вновь может войти в состояние Master. Далее эта последовательность срабатываний вершин повторяется. Утверждение доказано.

Утверждение 2. Пусть изначально все вершины графа системы находятся в состоянии Free и $n(i)$ – число срабатываний вершины i в состоянии Master в некоторый момент времени. Тогда для любых двух вершин $i, j \in \in \{0,1,\ldots,|V|-1\}$ графа системы выполняется

$$|n(i) - n(j)| \le D,$$

где D – диаметр графа ВС.

Доказательство. Рассмотрим произвольный путь на графе ВС как последовательность вершин $i_0, i_1, \ldots, i_L$, L – длина пути (количество ребер пути), $L \le D$, D – диаметр графа. Пусть номера вершин на этом пути упорядочены по возрастанию: $i_0 < i_1 < \ldots < i_D$. Для любой пары соседних вершин i_k, i_{k+1}, $k \in \{0,1,\ldots,D-1\}$, справедливо равенство $n(i_k) - n(i_{k+1}) = 1$, поскольку вершина i_k, завершив работу в состоянии Master, вновь сможет войти в это состояние только после того, как в состоянии Master сработает вершина i_{k+1}. Отсюда получаем

$$n(i_0) - n(i_L) = L \le D.$$

Из произвольности выбранного пути следует вышеприведенное утверждение.

Доказанное утверждение обосновывает асимптотическую корректность предложенного алгоритма в смысле соотношения

$$\lim_{n \to \infty} \frac{|n(i) - n(j)|}{n} = 0$$

для любых двух вершин $i, j \in \{0,1,\ldots,|V|-1\}$; $n = \min(n(i), n(j))$.

На рис. 2, *а* приведен пример ВС со структурой типа решетка. Если каждый домен пометить идентификатором его Master-вершины, то одна из возможных последовательностей срабатывания доменов (домены, идентификаторы которых заключены в скобки, срабатывают одновременно) с радиусом окрестности 1 на решетке (см. рис. 2, *а*) имеет вид

0, 2, 5, (1, 7, 8), (3, 4, 10), (0, 6, 13), (2, 9, 15), (5, 11, 12), (1, 7, 8, 14), (3, 4, 10), …

Далее последовательность срабатывания доменов циклически повторяется. На рис. 2, *b* – *f* представлен период данной последовательности разбиений ВС на домены. Как видно из этих рисунков, каждая из машин в цикле

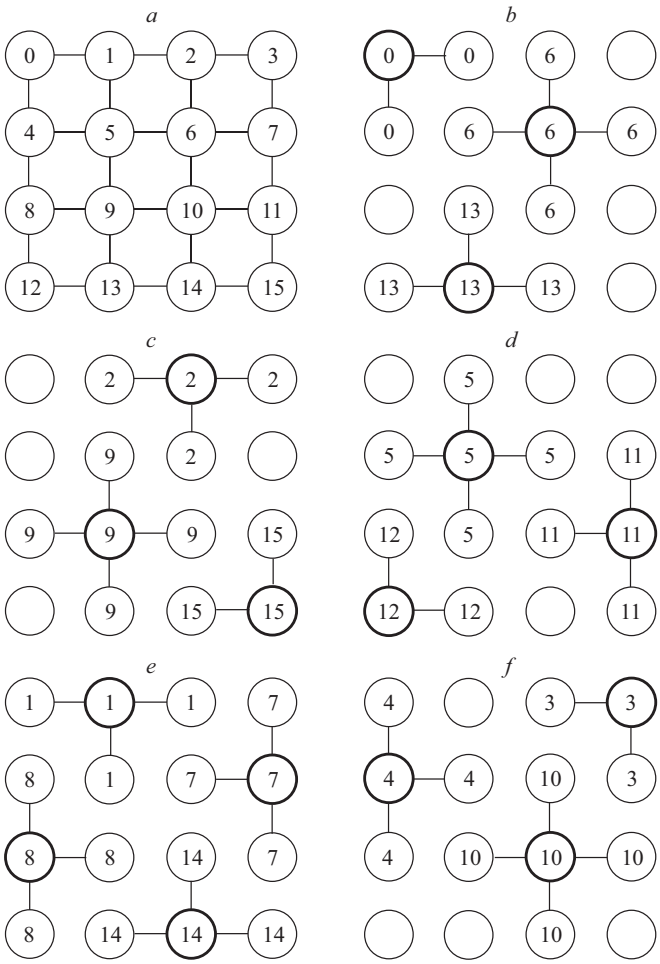

Рис. 2. Срабатывания доменов на решетке

срабатывает в состоянии Master 1 раз (машина в состоянии Master выделена жирной окружностью).

Предложенный подход к построению децентрализованных алгоритмов планирования позволяет независимо реализовать оптимизацию на доменах при условии, что глобальный критерий качества оптимизации можно свести к совокупности локальных критериев, каждый из которых оптимизируется на своем домене. Для подведения итогов децентрализованной оптимизации используется корневое восходящее дерево процессов, покрывающее граф ВС [4–6]. Каждый из этих процессов по достижении экстремума локального критерия и получении сообщений от всех дочерних процессов посылает родительскому процессу сообщение. Корневой процесс дерева делает вывод о завершении процесса оптимизации.

Рассмотрим применение предложенного подхода к построению децентрализованных алгоритмов планирования на примерах: 1) оптимального разбиения ВС на подсистемы для решения набора параллельных задач, 2) вложения структуры параллельной программы в структуру вычислительной системы (или подсистемы).

Оптимизация разбиения вычислительной системы на подсистемы для решения набора параллельных задач. При коллективном (многопользовательском или мультипрограммном) использовании ресурсов распределенной ВС с программируемой структурой [4] возникает необходимость представления множества ЭМ в виде объединения непересекающихся связных подмножеств (подсистем). Требуется распределить элементарные машины ВС, состоящей из N ЭМ, между $n < N$ непересекающимися подсистемами таким образом, чтобы удовлетворялся заданный критерий качества разбиения Ф.

Сформулируем критерий Ф следующим образом. Целесообразно обеспечить в каждой подсистеме максимум числа межмашинных соединений для достижения максимальной пропускной способности сети связи, что эквивалентно минимизации числа линий связи, пересекающих границы между подсистемами. Пусть E_{ij} – множество межмашинных линий связи на границе между подсистемами S_i и S_j, $i \neq j$. Тогда задачу оптимального разбиения можно представить в виде

$$\min \Phi = \frac{1}{2} \sum_{i,\ j=1}^{n} \left| E_{ij} \right|; \qquad \sum_{i=1}^{n} r_i = N, \tag{1}$$

где r_i – ранг (число ЭМ) подсистемы S_i. Значение Ф равно числу линий связи, не вошедших ни в одну из подсистем.

Каждой машине ВС случайным образом поставим в соответствие метку $i \in \{0,1,\ldots,n-1\}$ подсистемы, к которой эта машина изначально отнесена. Далее выполняется децентрализованный алгоритм, минимизирующий критерий (1), причем под управлением АСД в каждом из доменов реализуется локальный алгоритм оптимизации.

Выполнение локальной оптимизации сводится к перестановке меток $i \in \{0,1,\ldots,n-1\}$ машин в пределах домена. Локальный критерий оптимизации для задачи разбиения ВС на подсистемы имеет следующий вид:

$$\Phi_j = \sum_{k \in D_j} E_k, \qquad j = 0,1,\ldots,N-1,$$

где E_k – число ребер графа системы G_s, соединяющих машину с меткой k с машинами, имеющими метки $l \neq k$; D_j – домен с ведущей вершиной j.

На рис. 3, *а* приведен пример случайной начальной разметки вершин графа ВС для решения задачи оптимального разбиения на две подсистемы (бисекции системы). Для наглядности выбрана ВС со структурой типа решетка. На рис. 3, *б – е* представлены шаги вышеописанного алгоритма разбиения для доменов, соответствующих доменам на рис. 2, *б – е*. Из рис. 3, *е* следует, что получено оптимальное разбиение ВС на две подсистемы с минимальным числом граничных ребер между подсистемами. Естественно, результирующее разбиение не обязательно будет оптимальным при произвольной начальной разметке вершин в силу локальности их перестановок (любая перестановка выполняется в пределах домена) и локальности критерия оптимизации. Так, например, для случайной разметки на рис. 4, *а* последовательность срабатываний доменов, показанная на рис. 2, *б – f*, дает субоптимальное разбиение на рис. 4, *б*. Для получения глобального оптимума требуется

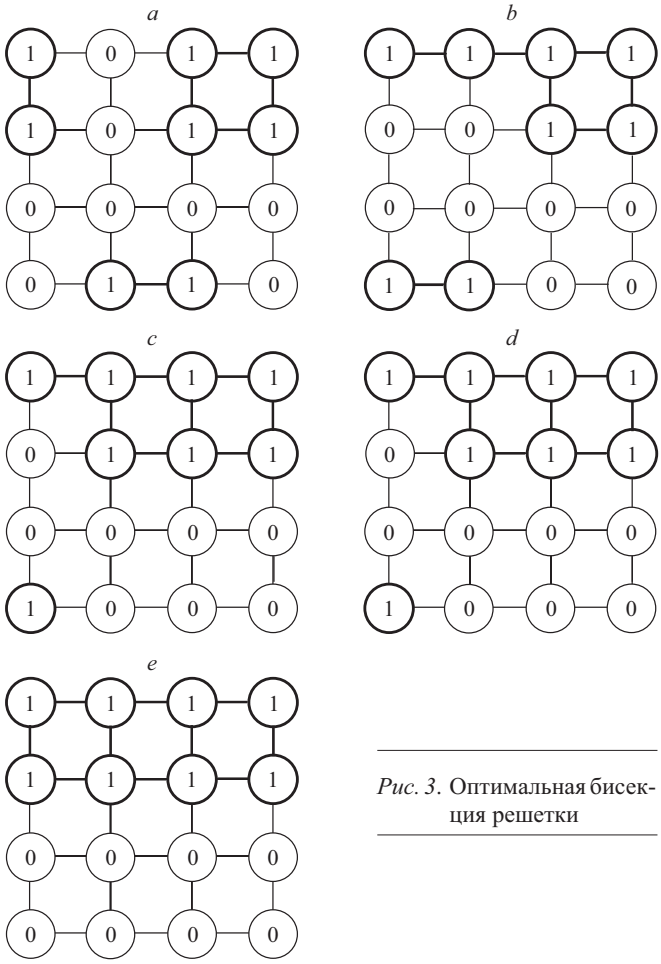

Рис. 3. Оптимальная бисекция решетки

перебор случайных начальных разметок с их последующей обработкой вышепредложенным алгоритмом.

Вложение структуры параллельной программы в структуру вычислительной системы. Для распределенных ВС граф $G_p(V_p, E_p)$ параллельной программы обычно определяется как множество V_p ветвей программы (виртуальных элементарных машин), взаимодействующих друг с другом

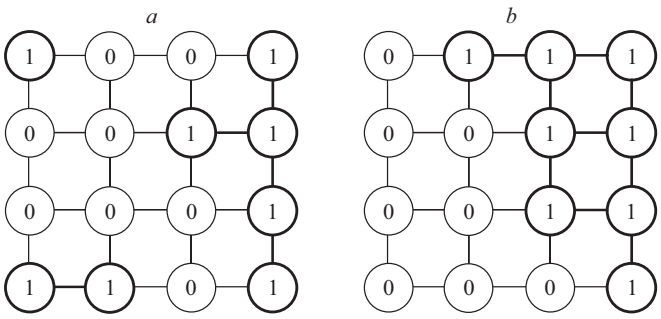

Рис. 4. Субоптимальная бисекция решетки

по принципу «точка–точка» посредством передачи сообщений по логическим (виртуальным) каналам (одно- и двунаправленным) множества $E_p \subseteq V_p \times V_p$. В общем случае вершинам $x, y \in V_p$ и ребрам (или дугам) $(x, y) \in E_p$ поставлены в соответствие числа (веса), характеризующие вычислительные сложности ветвей и интенсивности взаимодействий между ними. Во многих случаях, наблюдаемых в практике параллельного программирования, веса всех вершин (и всех ребер) графа программы можно считать одинаковыми. В этом случае задача вложения структуры параллельной программы в структуру распределенной ВС имеет следующий вид [5].

Граф $G_p(V_p, E_p)$ параллельной программы рассматривается как множество V_p вершин (ветвей программы) и функция

$$G_p\colon V_p \times V_p \to \{0, 1\},$$

удовлетворяющая

$$G_p(x, y) = G_p(y, x), \qquad G_p(x, x) = 0$$

для любых $x, y \in V_p$. Выражение $G_p(x, y) = 1$ означает, что существует ребро между вершинами x и y, т. е. $(x, y) \in E_p$. Аналогично граф $G_s = (V_s, E_s)$ определяется как множество вершин (ЭМ) V_s и функция

$$G_s\colon V_s \times V_s \to \{0, 1\}.$$

Здесь E_s – множество ребер (линий связи между ЭМ).

Пусть $|V_p| = |V_s| = n$. Обозначим вложение ветвей параллельной программы в ЭМ как одно-однозначную функцию

$$f_m\colon V_p \to V_s.$$

Качество вложения можно определить как число ребер графа программы, совпавших с ребрами графа ВС. Назовем это число мощностью $|f_m|$ вложения f_m и определим следующим выражением [5]:

$$|f_m| = (1/2) \sum_{x \in V_p,\ y \in V_p} G_p(x, y) G_s(f_m(x), f_m(y)). \qquad (2)$$

Согласно (2) критерий $|f_m|$ равен количеству ребер графа G_p программы, совпадающих с ребрами графа G_s вычислительной системы.

Как и в задаче разбиения ВС на подсистемы, оптимизация глобального критерия (2) сводится к оптимизации локальных критериев на доменах. Локальный критерий оптимизации, соответствующий глобальному критерию (2), имеет вид

$$|f_m|(\overline{D}_j) = (1/2) \sum_{x \in V_p(\overline{D}_j),\ y \in V_p(\overline{D}_j)} G_p(x, y) G_s(f_m(x), f_m(y)), \qquad (3)$$

где $\overline{D}_j$ – связный подграф, образуемый путем присоединения к домену D_j граничных с ним вершин; $V_p(\overline{D}_j)$ – множество вершин графа программы, отображенных в вершины домена $\overline{D}_j$.

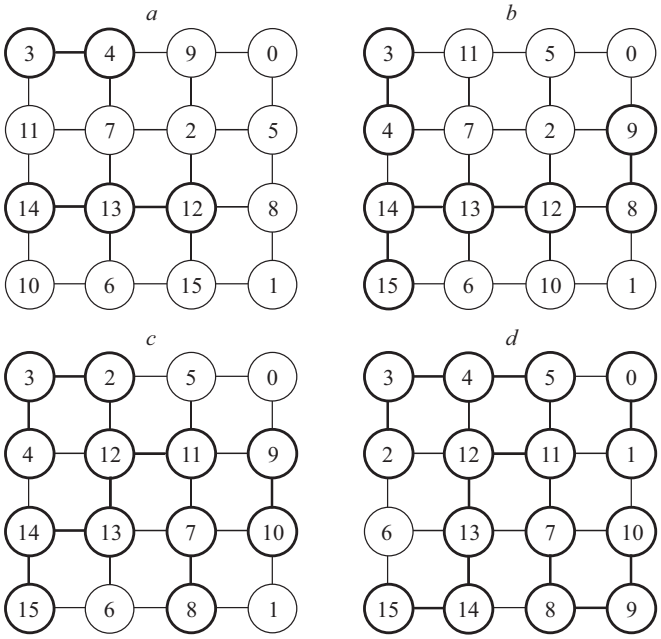

Рис. 5. Субоптимальное вложение линейки в решетку

На рис. 5, *a* приведен пример случайного начального отображения вершин графа программы в вершины графа вычислительной системы типа решетка. На рис. 5, *b* – *d* представлены результаты работы децентрализованного алгоритма, максимизирующего критерий (3), на доменах, соответствующих доменам на рис. 2, *b* – *d*. На рисунке жирными отрезками выделены ребра графа задачи, совпадающие с ребрами графа системы. Каждый шаг алгоритма оптимизации отображения сводится к перестановке пары вершин графа программы на домене при условии, что эта перестановка не ухудшает (не уменьшает) критерий отображения. Отображение рис. 5, *d* соответствует локальному максимуму критерия (3). Последующие шаги локальной оптимизации не улучшают значения критерия оптимизации, поэтому с целью поиска глобального минимума следует многократно повторить случайное начальное отображение с дальнейшей оптимизацией по описанному алгоритму.

Примеры эффективного применения предложенного подхода к конструированию децентрализованных алгоритмов управления ресурсами и заданиями (см. рис. 2–5) получены путем программного моделирования вышеописанных алгоритмов. В этих примерах допускались произвольные перестановки меток вершин в домене, не ухудшающие локального критерия оптимизации, что в общем случае, строго говоря, может на отдельных шагах привести к ухудшению глобального критерия. Для обеспечения монотонной сходимости оптимизационного процесса следует фиксировать метки на границах доменов, но это можно делать только при радиусе домена больше единицы.

Заключение. Итак, предложенный подход позволяет организовать согласованное одновременное выполнение множества перестановочных операций на структурах данных, распределенных по машинам вычислительной системы, и может служить основой для разработки новых эффективных де-

централизованных алгоритмов управления ресурсами и заданиями в живучих распределенных вычислительных системах.

СПИСОК ЛИТЕРАТУРЫ

1. **Корнеев В. В.** Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985.
2. **Димитриев Ю. К.** Самодиагностика модульных вычислительных систем. Новосибирск: Наука, 1994.
3. **Монахов О. Г., Монахова Э. А.** Параллельные системы с распределенной памятью: управление ресурсами и заданиями. Новосибирск: ИВМ и МГ СО РАН, 2001.
4. **Тарков М. С.** Параллельный алгоритм бисекции структуры распределенной вычислительной системы // Тр. VI Междунар. сем. «Распределенная обработка информации». Новосибирск, 1998. С. 96.
5. **Тарков М. С.** Вложение структур параллельных программ в структуры живучих распределенных вычислительных систем // Автометрия. 2003. **39**, № 3. С. 84.
6. **Тарков М. С.** Самостабилизация покрывающего дерева в макроструктуре распределенной вычислительной системы // Автометрия. 2001. № 2. С. 66.
7. **Tarkov M. S., Tsarenko A. V.** Data input algorithm for image processing in a distributed computer system // Pattern Recogn. and Image Analysis. 2001. **11**, N 2. P. 381.
8. **Ахо А., Хопкрофт Дж., Ульман Дж.** Построение и анализ вычислительных алгоритмов. М.: Мир, 1979.

Институт физики полупроводников СО РАН,
E-mail: tarkov@isp.nsc.ru

Поступила в редакцию
24 сентября 2004 г.