РОССИЙСКАЯ АКАДЕМИЯ НАУК
СИБИРСКОЕ ОТДЕЛЕНИЕ
АВТОМЕТРИЯ
2003, том 39, № 3

УДК 681.3.06 : 681.323

М. С. Тарков
(Новосибирск)

ВЛОЖЕНИЕ СТРУКТУР ПАРАЛЛЕЛЬНЫХ ПРОГРАММ В СТРУКТУРЫ ЖИВУЧИХ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ*

Предложен метод вложения структур параллельных программ в структуры живучих распределенных вычислительных систем (ВС). Разработаны эффективные алгоритмы для реализации этапов метода: 1) эвристический алгоритм отображения вершин графа параллельной программы в граф распределенной ВС, существенно сокращающий время вложения по сравнению с известным алгоритмом Бохари; 2) децентрализованный алгоритм отображения ребер графа программы, не совпадающих с ребрами графа ВС, в кратчайшие пути на графе ВС. Исследовано вложение одномерных (линейка, кольцо) и двумерных (решетка, тор) структур параллельных программ в регулярные структуры (тор, двумерный циркулянт, гиперкуб) живучих вычислительных систем с неисправными компонентами (машинами и межмашинными соединениями). Показано, что одномерные структуры параллельных программ вкладываются в структуры распределенных ВС лучше, чем двумерные, и при возникновении дефектов в структуре ВС (отказов компонентов ВС) качество вложения одномерных структур ухудшается меньше, чем качество вложения двумерных структур.

Введение. Распределенная вычислительная система (ВС) [1–4] представляет собой множество элементарных машин (ЭМ), связанных сетью, программно управляемой этими машинами. Каждая элементарная машина включает вычислительный модуль (ВМ) и системное устройство (СУ). Системное устройство функционирует под управлением ВМ и имеет входные и выходные полюсы, связанные соответственно с выходными и входными полюсами соседних ЭМ. Структура ВС описывается графом $G_s(V_s, E_s)$, где V_s – множество ЭМ, а $E_s \subseteq V_s \times V_s$ – множество связей между ЭМ. Для распределенных ВС граф $G_p(V_p, E_p)$ параллельной программы обычно определяется как множество V_p ветвей программы (виртуальных элементарных машин), которые взаимодействуют друг с другом 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

* Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект № 01-01-00790).

(веса), характеризующие вычислительные сложности ветвей и интенсивности взаимодействий между ними.

Задача вложения структуры параллельной программы в структуру распределенной ВС эквивалентна *NP*-полной задаче изоморфизма графов [5, 6]. Усилия исследователей направлены на поиск эффективных эвристик, пригодных для большинства ситуаций. 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 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

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

для любых $x, y \in V_p$. Равенство $G_p(x, y) = 1$ 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 (элементарных машин) 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 его как критерий-максимум качества вложения [5]:

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

Согласно (1) критерий $|f_m|$ равен количеству ребер графа G_p программы, совпадающих с ребрами графа G_s вычислительной системы.

Критерий-минимум качества вложения имеет вид

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

Здесь $L_s(f_m(x), f_m(y))$ равно расстоянию между вершинами $f_m(x)$ и $f_m(y)$ на графе G_s.

Общая схема алгоритма вложения. В вышеприведенной постановке вложение графа программы $G_p(V_p, E_p)$ в граф ВС $G_s(V_s, E_s)$ сводится к отображению f_m вершин графа G_p в вершины графа G_s (в соответствии с крите-

рием (1) или (2)) и вложению ребер (x, y) графа $G_p(V_p, E_p)$, не совпадающих с ребрами графа $G_s(V_s, E_s)$, в пути на графе $G_s(V_s, E_s)$ между вершинами $f_m(x)$ и $f_m(y)$.

Известные алгоритмы [5, 6] поиска оптимального вложения f_m в виде [5] сводятся к заданию начального вложения графа программы G_p в граф ВС G_s и последующим итеративным перестановкам вершин графа G_p с целью достижения экстремума критерия качества вложения. Основным недостатком этих алгоритмов является их сложность, вызванная необходимостью анализа $O(n^2)$ перестановок на каждой итерации алгоритма. При децентрализованной реализации этого подхода сложность алгоритма вложения усугубляется необходимостью выполнения множества взаимодействий между вершинами графа G_s (элементарными машинами ВС). Затраты на организацию взаимодействий можно существенно снизить, если вложение f_m реализовать следующей процедурой.

1. Собрать в произвольной машине ЭМ_0 описание графа G_s, используя информацию каждой ЭМ о ее собственных соседях.

2. Вычислить в ЭМ_0 вложение f_m.

3. Передать в каждую ЭМ системы ее номер, задаваемый вложением f_m.

4. Отобразить ребра (x, y) графа $G_p(V_p, E_p)$, не совпадающие с ребрами графа $G_s(V_s, E_s)$, в кратчайшие пути на графе $G_s(V_s, E_s)$ между вершинами $f_m(x)$ и $f_m(y)$.

Инициализация и завершение вложения. Для инициализации и завершения алгоритма вложения необходимо построить виртуальную систему взаимодействующих процессов (виртуальных вершин) со структурой типа покрывающее корневое дерево и одно-однозначным соответствием между процессами и машинами ВС. Такая организация взаимодействия процессов обусловлена следующими причинами:

1) покрывающее дерево может быть реализовано на произвольном связном графе ВС;

2) децентрализованный алгоритм построения корневого дерева, покрывающего граф ВС, прост и не является переборным. Каждая вершина дерева получает: свой идентификатор, расстояние до корня дерева, вес поддерева с корнем в данной вершине.

Вес поддерева равен числу вершин поддерева и ставится в соответствие входной дуге его корневой вершины на нисходящем дереве. В алгоритме построения дерева используются следующие обозначения: v – степень вершины; q – номер входящей (исходящей) дуги на нисходящем (восходящем) корневом дереве; $s(p)$ – признак наличия вершины, соседней с данной по ребру (дуге) p, $s(p)=1$, если сосед существует, иначе $s(p)=0$; $w(p)$ – вес ребра p, изначально $w(p)=0$; E_t – идентификатор вершины в дереве t; N_t – число вершин в дереве t; D_t – расстояние от данной вершины до корня дерева t; $\text{wait}(q, \text{mesg})$ – процедура, выполняющая циклический опрос входных дуг p, $p=0,1,\ldots,\text{v}-1$, пока сообщение mesg не будет получено по одной из дуг $q \in \{0,1,\ldots,\text{v}-1\}$; $\text{send}(p, \text{mesg})$ – неблокирующая процедура передачи сообщения mesg по дуге p; $\text{receive}(p, \text{mesg})$ – блокирующая процедура получения сообщения mesg по дуге p; root – признак того, что данная вершина является корнем дерева, $\text{root}=1$, если данная вершина – корень, иначе $\text{root}=0$. В качестве корневой вершины (лидера) может быть выбрана, например, вершина, через которую осуществляется ввод информации в систему [7].

Алгоритм построения покрывающего дерева имеет вид

```
Procedure Spanning_Tree:
begin
  mesg = 0;
if (root = 0) wait (q, mesg); else q = v;
      for p = 0 to v – 1 do
if s(p) <> 0 and p <> q
   send (p, mesg);
   mesg := 1; for p = 0 to v – 1 do
                begin if s(p) <> 0 and p <> q receive(p, w(p)); mesg := mesg + w(p) end;
  if root = 1 begin E_t := 0; N_t := mesg; D_t := 0 end;
     else begin send (q, mesg); receive(q, E_t); receive(q, N_t); receive(q, D_t) end;
  mesg := E_t + 1;
  for p = 0 to v – 1 do
    if w(p) <> 0

 begin send (p, mesg); mesg := mesg + w(p); send (p, N_t); send (p, D_t + 1) end;

       end;
```

Данный алгоритм порождает вдвое меньшее количество сообщений, чем алгоритм, предложенный в [8], поскольку не требует передачи ответа на сообщение. Роль этого ответа играет сообщение, посланное в такую вершину соседом.

Построенное покрывающее дерево используется для определения момента завершения выполнения алгоритма вложения. Завершение выполняется за два шага. На первом шаге каждая вершина: 1) получает от предшественников на восходящем дереве сообщения о завершении ими алгоритма поиска соседей (алгоритма построения кратчайших путей); 2) посылает (после завершения собственного алгоритма поиска соседей) преемнику на восходящем дереве сообщение о завершении алгоритма поиска для всех вершин поддерева с корнем в данной вершине. На втором шаге каждая вершина получает от своего предшественника на нисходящем дереве разрешение на выполнение вложенной параллельной программы пользователя. После получения разрешения вершина транслирует его своим преемникам на нисходящем дереве и инициирует исполнение вложенной параллельной программы.

В работе [9] предложен алгоритм, поддерживающий покрывающее корневое дерево при произвольных отказах вершин и ребер графа ВС, не нарушающих связности графа.

Вложение вершин графа программы. Рассмотрим следующий подход к задаче вложения [5]. Пусть некоторым образом задано начальное вложение вершин графа программы в вершины графа ВС, например, $f_m(x) = x$, т. е. номера ветвей графа совпадают с номерами содержащих эти ветви машин. Пусть $e_p(x)$ – окружение (множество соседей) вершины x на графе G_p и $e_s(x)$ – ее окружение на графе G_s. Для каждой вершины $x \in V_p$ протестируем перестановку вершин i и j, $i \neq j$, удовлетворяющих условию

$$i \in e_p(x) \,\&\, i \notin e_s(x) \,\&\, j \in e_s(x) \,\&\, \text{state}(j) = 0.$$

Условие state(j) = 0 означает, что вершина j еще не подвергалась перестановке в окружении $e_s(x)$, в противном случае state(j) = 1. Если перестановка не ухудшает качество вложения f_m, мы ее фиксируем. Такой подход основан на предположении о высокой вероятности ситуации, когда такая перестановка, увеличивающая количество вершин $i \in e_p(x)$ в окружении $e_s(x)$, улучшит (или по крайней мере не ухудшит) значение критерия качества $|f_m|$. Число тестируемых перестановок при одноразовом обходе всех вершин $x \in V_p$, очевидно, не превышает значения $v_p v_s n$, $n = |V_p|$, где v_p и v_s – максимальные степени вершин графов G_p и G_s соответственно. При $v_p v_s < n$ такой подход обеспечит сокращение объема вычислений по сравнению с известным алгоритмом Бохари (B-алгоритмом) [5], и при увеличении n эффект сокращения возрастает. На основе предложенного здесь подхода разработана процедура Search поиска локального экстремума функции $|f_m|$ в алгоритме вложения (MB-алгоритме), являющемся модификацией B-алгоритма.

Кроме вышеописанных, MB-алгоритм содержит следующие отличия от B-алгоритма. Во-первых, начальная величина $|f_m|$ сравнивается с E_p, и если равенство выполняется, то алгоритм завершается. Во-вторых, проверка равенства $|f_m| = |E_p|$ выполняется после каждой процедуры Search. Такие проверки зачастую приводят к сокращению времени выполнения алгоритма.

Далее представлен алгоритм для критерия (1).

```
Procedure MB;
begin
  done = false: BEST ← VS;
    if (card(VS) = card(TASK)) done = true;
    while (done ≠ true) do
      begin Search(VS, TASK);
        if (card(VS) > card(BEST))
          begin BEST ← VS: Rand_exchange(VS) end;
        else done = true;
      end;
    Output(BEST)
end.
```

Здесь TASK – описание G_p; VS – текущее вложение f_m; BEST – лучшее найденное вложение f_m; card(f) – значение критерия для вложения f; card(TASK) – значение критерия для вложения графа G_p в себя; BEST ← VS – создание копии BEST вложения VS; Rand_exchange(VS) – случайная перестановка n пар вершин во вложении VS; Output(BEST) – вывод наилучшего вложения f_m.

Исследовано качество MB-алгоритма при вложении типовых структур (линейка, кольцо, решетка) параллельных программ (рис. 1) в регулярные графы распределенных ВС (E_2-граф (тор), оптимальный D_2-граф (циркулянт), гиперкуб) с числом вершин n от 8 до 1024.

В d-мерном гиперкубе H_d с 2^d вершинами две вершины i и j связаны, если и только если расстояние Хемминга между ними $H(i, j) = 1$ (рис. 2, a, гиперкуб H_3). Циркулянт $D_n(N, l_1, ..., l_n)$ – это n-мерный граф с N вершинами, где вершины i и j связаны, если и только если $i - j = l_k \bmod N$, $k \in \{1, ..., n\}$, l_k – целые числа (рис. 2, b, циркулянт $D_2(16; 1,6)$). Тор $E_n(k_1, ..., k_n)$ с n измерениями имеет $N = k_1 \times k_2 \times ... \times k_n$ вершин, где вершины i и j связаны, если и только если $\exists l \in \{1, ..., n\}$, $(i - j) \bmod k_l = 1$ (рис. 2, c, тор $E_2(4, 4)$).

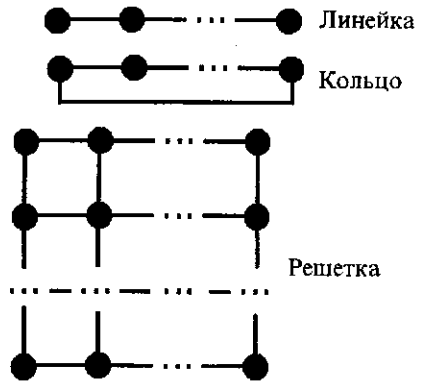

Рис. 1. Типовые графы параллельных программ

В табл. 1 представлены времена t_n выполнения МВ-алгоритма в секундах для некоторых из рассмотренных выше случаев вложения при изменении n от 8 до 1024 (алгоритм тестировался на персональном компьютере с процессором Intel Pentium III с тактовой частотой 800 МГц). Результаты тестирования МВ-алгоритма показали, что достигаемое алгоритмом вложение графа G_p в граф G_s с числом вершин $n = |V_p| = |V_s| \le 1024$ значение критерия (1) не меньше, чем 90 % от числа ребер графа G_p, если G_p – линейка или кольцо (табл. 2), и не меньше, чем 80 %, если G_p – решетка (табл. 3). В E_2-графы решетка вкладывается стопроцентно. Также стопроцентно вкладывается кольцо в тор $E_2(2k, m)$ с произвольным числом вершин $n = 2k \times m$.

МВ-алгоритм существенно снижает время вложения в сравнении с В-алгоритмом, особенно при вложении одномерных графов (линейка, кольцо) в E_2- и D_2-графы. При этом МВ-алгоритм не ухудшает качество вложения по отношению к В-алгоритму.

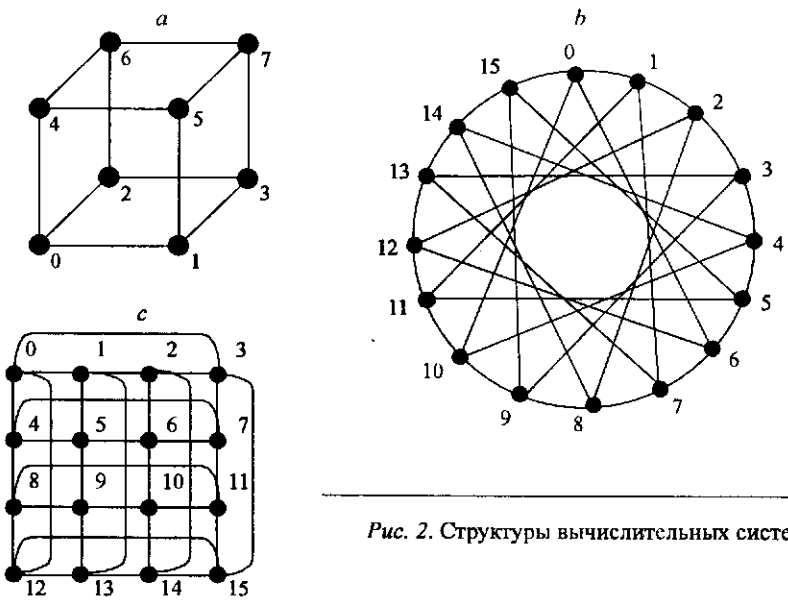

Рис. 2. Структуры вычислительных систем

Таблица 1

Время вложения типовых структур параллельных программ

Типы графов программ	E_2-граф (тор)		D_2-граф (циркулянт)		Гиперкуб	
	t_{64}	t_{1024}	t_{64}	t_{1024}	t_{64}	t_{1024}
Линейка	0,065	11,1	0,071	19,9	0,137	200,0
Кольцо	0,017	3,76	0,070	20,1	0,210	414,2
Решетка	–	–	0,109	36,3	0,264	647,3

Вложение ребер графа программы. После завершения вложения вершин графа программы G_p для каждой пары вершин x, $y \in G_p$, удовлетворяющих уравнению $G_p(x, y) = 1$, необходимо решить задачу построения кратчайшего пути на графе G_s между несоседними вершинами $f_m(x)$ и $f_m(y)$ ($G_s(f_m(x), f_m(y)) \neq 1$). Обозначим через $\mathrm{path}(f_m(x), f_m(y))$ кратчайший путь из вершины $f_m(x)$ в вершину $f_m(y)$, который соответствует ребру $(x, y) \in G_p$. Для каждой вершины $z \in G_s$ перенумеруем инцидентные ей ребра. Построение пути сводится к поиску в G_s вершин $z \in \mathrm{path}(f_m(x), f_m(y))$ и заданию в каждой вершине z отображения T: $y \to s(z)$ номера y целевой вершины в номер $s(z)$ ребра $(z, z') \in G_s$, $(z, z') \in \mathrm{path}(f_m(x), f_m(y))$, z' – вершина, соседняя с z, т. е. $G_s(z, z') = 1$. Множество значений функции T: $y \to s(z)$ образует в вершине z таблицу маршрутизации сообщений. Пути из $f_m(x)$ в $f_m(y)$ будем строить одновременно для всех $x \in V_p$, $y \in e_p(x)$. Процедура поиска кратчайшего пути состоит из двух этапов.

На первом этапе выполним рассылку сообщений из вершины $f_m(x)$ по покрывающему дереву ВС (подсистемы ВС), в котором $f_m(x)$ является кор-

Таблица 2

Вложение кольца в тор

Алгоритмы вложения	Время вложения		Качество вложения	
	t_{64}	t_{256}	$\lvert f_{64}\rvert/\lvert E_{64}\rvert$	$\lvert f_{256}\rvert/\lvert E_{256}\rvert$
В-алгоритм	0,478	38,124	0,91	0,95
МВ-алгоритм	0,017	0,245	1,0	1,0

Таблица 3

Вложение решетки в гиперкуб

Алгоритмы вложения	Время вложения		Качество вложения	
	t_{64}	t_{256}	$\lvert f_{64}\rvert/\lvert E_{64}\rvert$	$\lvert f_{256}\rvert/\lvert E_{256}\rvert$
В-алгоритм	0,955	108,0	0,82	0,77
МВ-алгоритм	0,264	13,7	0,87	0,80

нем. Назовем вершину $f_m(x)$ ведущей. Ведущая вершина посылает сообщение ЗАПРОС(x) всем своим соседям. Если какая-либо вершина получает сообщение ЗАПРОС(x) впервые, то она транслирует полученное сообщение всем своим соседям за исключением того соседа, от которого это сообщение было получено. Сообщение ЗАПРОС(x), наряду с номером искомой вершины, содержит расстояние $d(x)$ от вершины z, которая получила сообщение, до ведущей вершины $f_m(x)$ (корневой вершины покрывающего дерева). После получения сообщения ЗАПРОС(x) вершина z увеличивает значение $d(x)$ на единицу. Если вершина получила несколько сообщений ЗАПРОС(x) (от нескольких соседей), то в качестве связи с предшественником на нисходящем дереве выбирается ребро, по которому было получено сообщение с минимальным значением $d(x)$. В результате множество ребер графа G_s с минимальными значениями $d(x)$ образуют дерево, покрывающее ВС (или подсистему ВС), с корнем в вершине $f_m(x)$. В этом дереве каждая вершина связана с корневой $f_m(x)$ кратчайшим путем. После первого получения сообщения ЗАПРОС(x) и его трансляции каждая вершина ожидает сообщений типа ЗАПРОС(x) или ОТВЕТ(x) от всех своих соседей, кроме тех, от которых сообщение было получено впервые. Сообщение ОТВЕТ(x) может содержать номера $f_m(y)$ вершин $y \in e_p(x)$.

Когда вершина получила сообщения от всех своих соседей, она, если это не $f_m(x)$ (т. е. не ведущая), формирует сообщение ОТВЕТ(x), содержащее все полученные номера $f_m(y)$ найденных вершин $y \in e_p(x)$ и собственный номер $f_m(z)$, если $z \in e_p(x)$. Далее сообщение ОТВЕТ(x) передается преемнику на восходящем к $f_m(x)$ дереве. Иначе, если данная вершина является ведущей (т. е. $f_m(x)$), то алгоритм поиска кратчайших путей из вершины $f_m(x)$ в вершины $f_m(y)$, $y \in e_p(x)$, завершен.

В данном алгоритме не используется задание временных интервалов для определения момента завершения алгоритма. Это достигается за счет рассылки избыточных сообщений. Независимо от качества отображения вершин, т. е. от расстояния между вершинами $f_m(x)$ и $f_m(y)$, $G_s(f_m(x), f_m(y)) \neq \neq 1$, граф передачи сообщений ОТВЕТ(x) образует дерево, покрывающее ВС (подсистему ВС).

Описанный алгоритм может быть реализован одновременно и независимо для всех вершин $f_m(x)$, $x \in V_p$, $f_m(x) \in V_s$, так как сообщения ЗАПРОС(x) и ОТВЕТ(x) помечены номером x ведущей вершины. Для поиска соседей вершины графа программы (построения кратчайших путей) необходимо выполнить $O(\log_{v-1} n)$ межмашинных обменов сообщениями, где n – число вершин графа и v – степень вершины. Таким образом, для поиска соседей всех вершин необходимо выполнить $O(n \log_{v-1} n)$ обменов. Так как одновременно можно выполнить не более чем vn обменов, время выполнения алгоритма равно $O(\log_{v-1} n)$. Для предотвращения тупика каждая ЭМ должна иметь оперативную память, достаточную для хранения сообщений. На каждом шаге алгоритма ЭМ получает v сообщений, но может транслировать только одно (в наихудшем случае). Поэтому сообщения могут накапливаться в ЭМ. Для хранения этих сообщений ЭМ должна иметь память размером $(v-1)O(\log_{v-1} n)V_{mes}$, где V_{mes} – размер памяти для хранения одного сообщения.

Условия вложения параллельной программы в живучую ВС с неисправными компонентами. Используя МВ-алгоритм, исследуем вложение структур параллельных программ в структуры распределенных ВС с неисправными компонентами. Качество вложения исследуется для различных

топологий ВС при наличии неисправных ЭМ или межмашинных соединений. Неисправность межмашинной связи означает отсутствие соответствующего ребра в графе ВС. Неисправность ЭМ означает отсутствие в графе ВС соответствующей вершины с инцидентными ей ребрами.

Обычно графы ВС изначально регулярны (тор, гиперкуб и т. д.), но при работе ВС возникают отказы ЭМ и межмашинных соединений. В результате параллельную программу приходится заново вкладывать в нерегулярную связную часть ВС. Децентрализованная операционная система ВС обеспечивает равенство числа ветвей параллельной программы числу функционирующих ЭМ, т. е. $|V_s| = |V_p|$ и вычислительная нагрузка равномерно распределяется между функционирующими ЭМ [3, 4]. Другими словами, параллельная программа может быть настроена на число функционирующих процессоров ВС.

При каком количестве неисправных компонентов вложение параллельной программы в вычислительную систему еще возможно? Имеется два типа ограничений на допустимую кратность отказов. Во-первых, количество неисправных вершин не может превышать $(n-1)/2$, $n = |V|$. Это ограничение определяется условием самодиагностируемости [2]. Согласно этому ограничению алгоритм самодиагностики работает правильно, если только исправ-

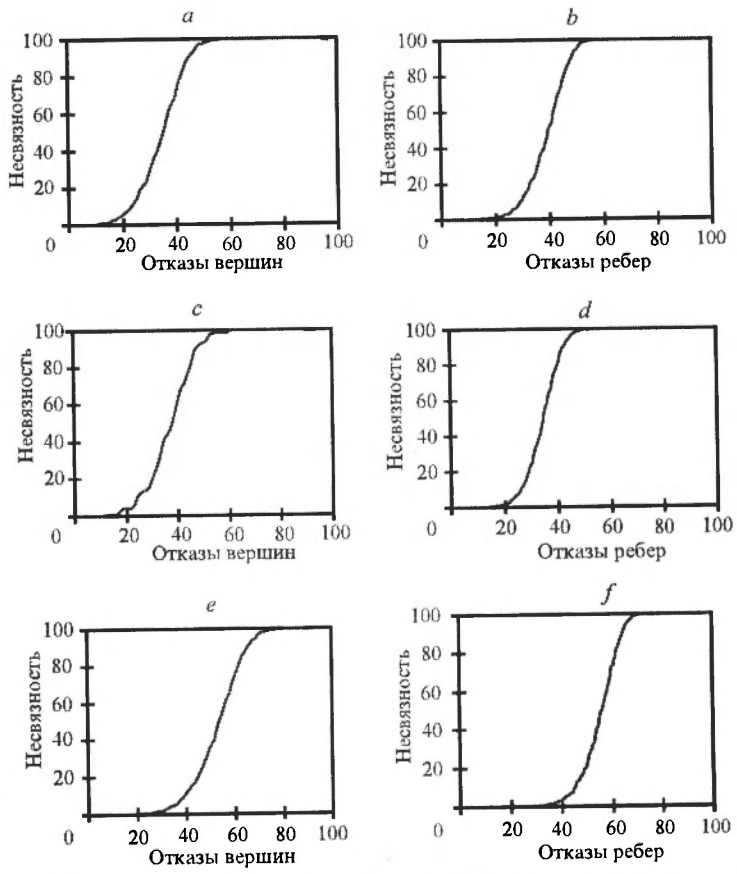

Рис. 3. Зависимости вероятности несвязности графа ВС от процента неисправных вершин и ребер

ная часть системы включает в себя не менее половины суммарного числа машин. Во-вторых, граф с исправными вершинами и ребрами должен быть связным. Это свойство необходимо для организации взаимодействий между ветвями параллельной программы. Вероятность p_t несвязности графа ВС и вероятность $p_c = 1 - p_t$ связности графа были исследованы в зависимости от числа неисправных вершин и ребер. Методом Монте-Карло проведено моделирование регулярных графов с неисправными вершинами и ребрами. Результаты моделирования приведены на рис. 3. Этот рисунок демонстрирует зависимости p_t от процента неисправных вершин (рис. 3, a, c, e) и ребер (рис. 3, b, d, f) для тора (см. рис. 3, a, b), циркулянта (см. рис. 3, c, d) и гиперкуба (булева куба) (см. рис. 3, e, f) с числом вершин $n = 64$.

Результаты моделирования показывают: 1) для торов и двумерных циркулянтов $p_t \approx 1$ при $t > n/2$, а для гиперкубов $p_t \approx 1$ при $t > 3n/4$; 2) для торов и двумерных циркулянтов $p_c \approx 1$ при $t \le 0{,}2n$ (см. рис. 3 , $a-d$), а для гиперкубов $p_c \approx 1$ при $t \le 0{,}3n$ (см. рис. 3, e, f).

Эти результаты справедливы как для графов с неисправными вершинами, так и для графов с неисправными ребрами. Во втором случае n – это число ребер ВС. Результаты показывают, что графы с большей степенью вершины допускают большее число отказов, не нарушающих связности. Таким образом, при кратности отказов $t \le 0{,}2n < (n-1)/2$ все рассмотренные топологии (типы графов) почти всегда являются связными и возможно применение вышеописанного алгоритма вложения.

Исследование алгоритма вложения. Рассмотрены четыре типа графов G_p параллельных программ: линейка, кольцо, решетка и тор. Эти графы с числом вершин от $n = 8$ до $n = 256$ отображались в графы ВС (тор, двумерный циркулянт (D_2-граф), гиперкуб) с неисправными вершинами или неисправными ребрами. План исследования алгоритма вложения имеет следующий вид.

1. Создать начальное описание графа ВС с заданной регулярной топологией и заданным числом вершин.

2. Задать кратность отказов t и породить множество описаний искаженных графов с t отказами вершин (или ребер) начального графа.

3. Вложить граф программы в каждый искаженный граф ВС и вычислить качество каждого вложения как отношение $|f_m|$ к общему числу вершин (или ребер) графа ВС.

4. Вычислить среднее значение r_m качества вложения. Результаты моделирования алгоритма вложения можно разбить на две группы: вложение ли-

Таблица 4

Минимальные значения показателя качества вложения

Типы графов программ	Тор		Циркулянт		Гиперкуб	
	$r_m(V)$	$r_m(E)$	$r_m(V)$	$r_m(E)$	$r_m(V)$	$r_m(E)$
Линейка	0,802	0,803	0,801	0,802	0,868	0,867
Кольцо	0,793	0,796	0,797	0,790	0,859	0,850
Решетка	0,571	0,792	0,538	0,790	0,633	0,613
Тор	0,500	0,595	0,495	0,468	0,586	0,586

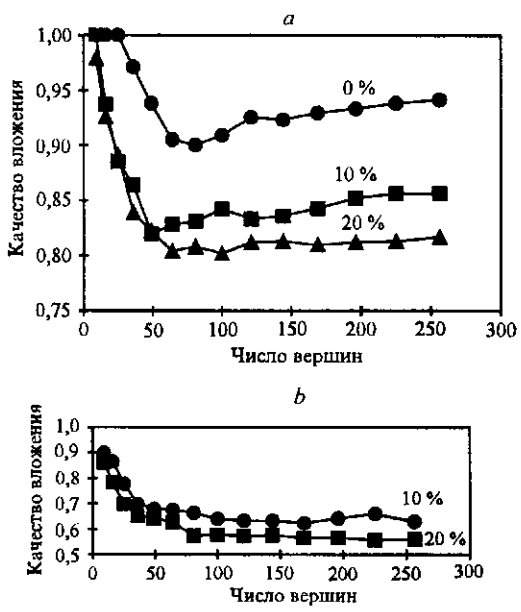

Рис. 4. Качество вложения структур программ в тор

нейки и кольца и вложение тора и решетки. В табл. 4 приведены минимумы усредненных значений $r_m(V)$ и $r_m(E)$ качества отображения при отказах вершин и ребер соответственно.

Для первой группы при числе неисправных вершин $t < 0{,}2n$ среднее значение качества вложения не ниже 0,8, если граф ВС имеет топологию тора или циркулянта, и не ниже 0,86, если граф ВС имеет топологию гиперкуба. Для второй группы при числе неисправных вершин $t < 0{,}2n$ среднее значение

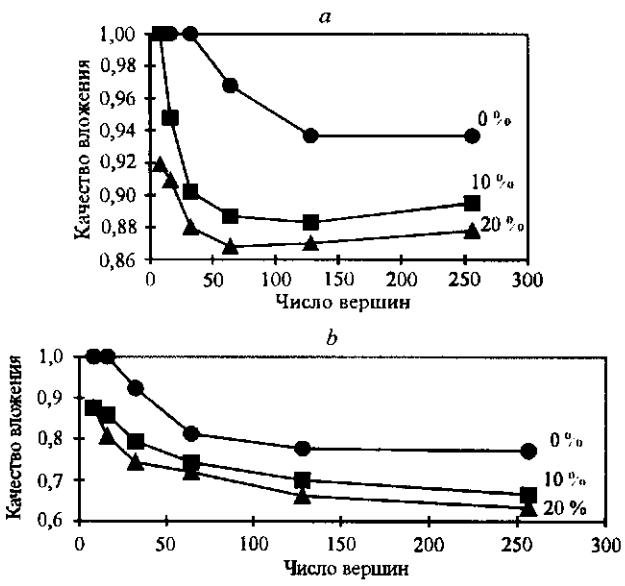

Рис. 5. Качество вложения структур программ в гиперкуб

Таблица 5

Вложение линейки в тор

Неисправные вершины (%)	Число вершин						
	16	36	64	100	144	169	225
0	1,0	0,971	0,905	0,909	0,923	0,929	0,938
10	0,937	0,864	0,828	0,842	0,836	0,843	0,856
20	0,926	0,839	0,804	0,802	0,813	0,810	0,813

Таблица 6

Вложение решетки в тор

Неисправные вершины (%)	Число вершин						
	16	36	64	100	144	196	256
10	0,865	0,698	0,674	0,639	0,633	0,641	0,628
20	0,783	0,652	0,626	0,577	0,575	0,566	0,558

Таблица 7

Вложение линейки в гиперкуб

Неисправные вершины (%)	Число вершин					
	8	16	32	64	128	256
0	1,0	1,0	1,0	0,968	0,937	0,937
10	1,0	0,948	0,902	0,887	0,883	0,895
20	0,919	0,909	0,880	0,868	0,870	0,878

Таблица 8

Вложение решетки в гиперкуб

Неисправные вершины (%)	Число вершин					
	8	16	32	64	128	256
0	1,0	1,0	0,923	0,812	0,776	0,771
10	0,875	0,857	0,794	0,743	0,699	0,665
20	0,875	0,805	0,742	0,719	0,661	0,633

качества вложения не ниже 0,46, если граф ВС имеет топологию тора или циркулянта, и не ниже 0,58, если граф ВС имеет топологию гиперкуба.

На рис. 4, 5 приведены графики зависимости r_m от числа n вершин графа ВС при некоторых значениях процента неисправных вершин (табл. 5–8). На рис. 4, a, b показаны зависимости качества вложения в тор линейки и решетки соответственно. На рис. 5, a, b представлены зависимости качества вложения в гиперкуб этих же графов программ. В табл. 5 и 6 приведены значения качества вложения в тор линейки и решетки соответственно, а в табл. 7 и 8 – значения качества вложения в гиперкуб этих же графов программ.

Заключение. Предложен метод вложения графов параллельных программ в графы живучих распределенных вычислительных систем. Разработаны эффективные алгоритмы для реализации этапов вложения: 1) эвристический алгоритм отображения вершин графа параллельной программы в вершины графа распределенной ВС, существенно сокращающий время вложения по сравнению с известным алгоритмом Бохари; 2) децентрализованный алгоритм отображения ребер графа программы, не совпадающих с ребрами графа ВС, в кратчайшие пути на графе ВС. Моделирование алгоритма отображения вершин показало, что качество вложения графов параллельных программ зависит от степеней их вершин: чем ниже степени вершин графа программы, тем качество вложения выше и при возникновении дефектов в графе ВС (исключений вершин и ребер, соответствующих отказам машин и межмашинных соединений ВС) ухудшается меньше.

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. **Корнеев В. В., Тарков М. С.** Операционная система микрокомпьютерной системы с программируемой структурой МИКРОС // Микропроцессорные средства и системы. 1988. **4**. С. 41.
4. **Тарков М. С.** Параллельная отказоустойчивая обработка изображений в транспьютерной системе МИКРОС-Т // Научное приборостроение.1995. **5**, № 3–4. С. 74.
5. **Bokhari S. H.** On the mapping problem // IEEE Trans. Comput. 1981. **C-30**, N 3. P. 207.
6. **Lee S.-Y., Aggarval J. K.** A mapping strategy for parallel processing // IEEE Trans. Comput. 1987. **C-36**, N 4. P. 433.
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. **Chang E.** Echo algorithms: Depth parallel operations on general graphs // IEEE Trans. Soft. Eng. 1982. **SE-8**, N 4. P. 391.
9. **Тарков М. С.** Самостабилизация покрывающего дерева в макроструктуре распределенной вычислительной системы // Автометрия. 2001. № 2. С. 66.

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

Поступила в редакцию
25 марта 2003 г.