АЛГОРИТМЫ ПАРАЛЛЕЛЬНОГО ПЕРЕЧИСЛЕНИЯ РАЗБИЕНИЙ МНОЖЕСТВА

В. В. Семенов

Томский государственный университет

Рассматриваются задача перечисления всевозможных разбиений n-элементного множества и задача перечисления всех разбиений n-элементного множества на k блоков. Эти задачи возникают, например, при построении бесповторных ДНФ булевых функций. Предлагаются два параллельных алгоритма решения первой задачи и параллельный алгоритм решения второй. Каждый алгоритм обеспечивает равномерную загрузку процессоров при условии полного отсутствия взаимодействия между ними.

В первом алгоритме, рассчитанном на применение m процессоров, i-й процессор ($i = 1, \ldots, m$) перечисляет все разбиения на k блоков для $k_{i-1} < k \le k_i$, где $0 = k_0 < k_1 < \ldots < k_m = n$. Разбиения перечисляются каждым процессором с помощью последовательного алгоритма, представленного в [1]. Экспериментальным путем доказано (см. табл. 1), что наиболее равномерная загрузка процессоров и, следовательно, максимальная эффективность их использования таким алгоритмом достигаются при $m = 4$, $k_1 = [n/2] - 2$, $k_2 = [n/2] - 1$, $k_3 = [n/2]$. Из табл. 2 видно, что 4-й процесс работает значительно дольше остальных.

Таблица 1

Время работы (в секундах) последовательного алгоритма перечисления k-блочных разбиений n-элементного множества

$n\backslash k$	3	4	5	6	7	8	9	10	11
12	0	15	15	31	16	0	0	0	0
13	0	46	156	187	125	47	0	0	0
14	31	203	859	1250	1000	469	140	32	0
15	63	828	4328	7969	7843	4484	1532	343	47

Таблица 2

Время работы каждого из четырех процессов в первом алгоритме для множеств мощности 14, 15 и 16

$m\backslash n$	14	15	16
1	1,6	13,9	83,5
2	2,7	16,5	163,5
3	2,8	29,6	227,7
4	5,9	51,6	275,6

Во втором алгоритме – параллельного перечисления всевозможных разбиений n-элементного множества – используется дерево разбиений (рис. 1), в котором вершины i-го яруса, $i = 1, \ldots, n$ представляют все разбиения i-элементного множества. Потомки вершины v (вершины следующего яруса, смежные с v), сопоставленной разбиению R, получаются либо добавлением нового элемента x в один из блоков разбиения R, либо добавлением в R блока $\{x\}$.

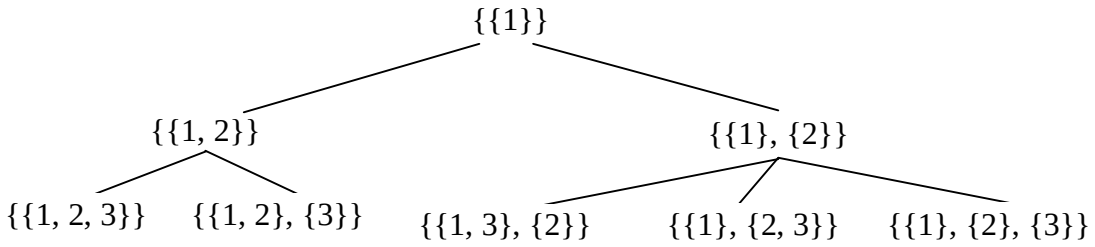


Рис. 1. Дерево разбиений множества $A = \{1, 2, 3\}$

Для параллельного перечисления разбиений используется параллельный обход дерева разбиений. На вход алгоритма подаются разбиваемое множество и номер яруса (*яруса назначений*), начиная с которого процессы, запущенные на разных процессорах, будут выполнять обход различных поддеревьев в дереве разбиений. Множество вершин яруса назначений разбивается на классы по числу процессов. Каждый из процессов поочередно выполняет обход поддеревьев с корневыми вершинами в назначенном ему классе.

Опишем действия, исполняемые каждым из m процессов с номерами $0, 1, \ldots, m–1$. Пусть t – номер яруса назначений. По последовательному алгоритму процесс перечисляет и нумерует разбиения t-элементного множества числами $0, 1, 2, \ldots$ до тех пор, пока не получит разбиение, номер которого сравним с номером процесса по модулю m. Тогда процесс выполняет обход поддерева с корнем в достигнутой вершине. После чего продолжает генерировать разбиения t-элементного множества для получения следующей корневой вершины своего класса.

Третий алгоритм предназначен для параллельного перечисления k-блочных разбиений n-элементного множества $A = \{a_1, \ldots, a_n\}$. Он отличается от второго алгоритма лишь структурой используемого дерева (рис. 2). В этом дереве потомки вершины v i-го яруса, соответствующей разбиению $R = \{R_1, R_2, \ldots, R_t\}$ множества $\{a_1, \ldots, a_i\}$, строятся по следующим правилам.

1. Если $(n – i) = (k – t)$, то единственный потомок вершины v соответствует разбиению $\{R_1, R_2, \ldots, R_t, \{a_{i+1}\}\}$, содержащему элемент a_{i+1} в отдельном блоке.

2. Если $t = k$, то вершина v имеет t потомков, которые сопоставляются следующим разбиениям множества $A_{i+1}=\{a_1, \ldots, a_{i+1}\}$: $\{R_1 \cup \{a_{i+1}\}, R_2, \ldots, R_t\}$, $\{R_1, R_2 \cup \{a_{i+1}\}, \ldots, R_t\}$, …, $\{R_1, R_2, \ldots, R_t \cup \{a_{i+1}\}\}$.

3. В остальных случаях вершина v имеет $(t+1)$ потомков, сопоставленных следующим разбиениям множества A_{i+1}: $\{R_1 \cup \{a_{i+1}\}, R_2, \ldots, R_t\}$, $\{R_1, R_2 \cup \{a_{i+1}\}, \ldots, R_t\}$, …, $\{R_1, R_2, \ldots, R_t \cup \{a_{i+1}\}\}$, $\{R_1, R_2, \ldots, R_t, \{a_{i+1}\}\}$.

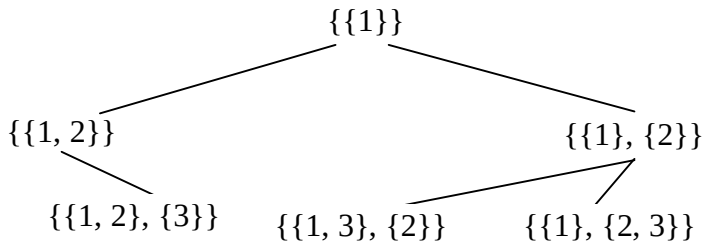


Рис. 2. Дерево разбиений множества
$A = \{1, 2, 3\}$ на 2 блока

Эффективность второго и третьего алгоритмов зависит от выбора номера яруса назначений. Очевидно, что чем больше номер яруса, тем меньшего размера поддеревья будут обрабатываться процессами и тем более равномерно будут загружены все процессоры. Однако если номер яруса взять слишком большим, то большая часть времени уйдет на построение корневых вершин, так как их построение ведется с помощью последовательного алгоритма, и, кроме того, каждый из процессов в итоге строит все разбиения, расположенные на ярусе назначений.

Таблица 3

Ускорение в алгоритме параллельного перечисления всевозможных разбиений *n*-элементного множества при *n* = 14, 15 и 16

n\s	2	3	4	5	6	7	8	9	10	11	12
14	1,99	2,99	3,99	4,96	5,95	6,96	7,91	8,96	9,88	10,84	11,79
15	1,99	2,99	3,98	4,95	5,93	6,96	7,87	8,95	9,85	10,86	11,74
16	1,99	2,98	3,97	4,92	5,92	6,95	7,83	8,93	9,80	10,84	11,68

Данные алгоритмы реализованы с помощью технологии *MPI* и экспериментально исследованы на кластерной системе с использованием от 2 до 14 процессоров. Для второго и третьего алгоритмов экспериментальным путем удалось подобрать такой номер яруса назначений, при котором эффективность распараллеливания близка к 1 (табл. 3, 4). Например, для множеств мощности n = 14, 15, 16 оптимальный номер яруса назначений равен 9 независимо от числа процессоров в системе.

Таблица 4

Ускорение в алгоритме параллельного перечисления *k*-блочных разбиений *n*-элементного множества при *n* = 15 и *k* = 4, 5, 6, 7 и 8

*k**s*	2	3	4	5	6	7	8	9	10	11	12
4	1,99	2,99	3,99	4,89	5,94	6,89	7,77	8,66	9,47	10,45	11,22
5	1,99	2,99	3,99	4,99	5,94	6,95	7,7	8,84	9,81	10,85	11,64
6	1,99	2,99	3,99	4,99	5,97	6,95	7,97	8,94	9,9	10,87	11,8
7	1,99	2,98	3,99	4,97	5,88	6,98	7,94	8,94	9,78	10,8	11,54
8	1,99	2,99	3,98	4,94	5,85	6,87	7,81	8,88	9,68	10,65	11,47

Достоинством представленных алгоритмов является отсутствие взаимодействия между процессорами, благодаря чему не тратится время на передачу данных, за исключением исходных. Кроме того, второй и третий алгоритмы являются хорошо масштабируемыми и позволяют достичь линейного ускорения.

Литература

1. Липский В. Комбинаторика для программистов. – М.: Мир, 1988. – 200 с.