3.3. ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ

Здесь рассматриваются основные понятия и приемы параллельного программирования, иллюстрируется методика крупноблочного распараллеливания сложных задач.

3.3.1. Элементарные понятия параллельного программирования

Понятие параллельного алгоритма относится к фундаментальным в теории вычислительных систем. Это понятие прежде всего ассоциируется с вычислительными системами с массовым параллелизмом. *Параллельный алгоритм – это описание процесса обработки информации, ориентированное на реализацию в коллективе вычислителей*. Такой алгоритм, в отличие от последовательного, предусматривает одновременное выполнение множества операций в пределах одного шага вычислений и как последовательный алгоритм сохраняет зависимость последующих этапов от результатов предыдущих.

Параллельный алгоритм решения общей задачи составляет основу параллельной программы P, которая, в свою очередь, влияет на алгоритм функционирования (см. 3.1.3) коллектива вычислителей. *Запись параллельного алгоритма на языке программирования,* доступном коллективу вычислителей, и *называют параллельной программой*, а сам язык – параллельным. Параллельные алгоритмы и программы следует разрабатывать для тех задач, которые недоступны для решения на средствах, основанных на модели вычислителя. Эти задачи принято называть сложными или трудоемкими.

Методы и алгоритмы обработки информации, решения задач, как правило, – последовательные. Процесс “приспособления” методов к реализации на коллективе вычислителей или процесс “расщепления” последовательных алгоритмов решения сложных задач называется *распараллеливанием (Paralleling or Multisequencing).*

Теоретическая и практическая деятельность по созданию параллельных алгоритмов и программ обработки информации называется *параллельным программированием* (*Parallel or Concurrent Programming*).

Качество параллельного алгоритма (или его эффективность) определяется методикой распараллеливания сложных задач. Существует два подхода при распараллеливании задач (рис. 3.3): *локальное* и *глобальное (крупноблочное) распараллеливание* [1]. Первый подход ориентирован на расщепление алгоритма решения сложной задачи на предельно простые блоки (операции или операторы) и требует выделения для каждого этапа вычислений максимально возможного количества одновременно выполняемых блоков. Он не приводит к параллельным алгоритмам, эффективно реализуемым коллективом вычислителей. В самом деле, процесс такого распараллеливания весьма трудоемок, а получаемые параллельные алгоритмы характеризуются не только структурной неоднородностью, но и существенно разными объемами операций на различных этапах вычислений. Последнее является серьезным препятствием на пути (автоматизации) распараллеливания и обеспечения эффективной эксплуатации ресурсов коллектива вычислителей. Локальное распараллеливание позволяет оценить предельные возможности коллектива вычислителей при решении сложных задач, получить предельные оценки по распараллеливанию сложных задач.

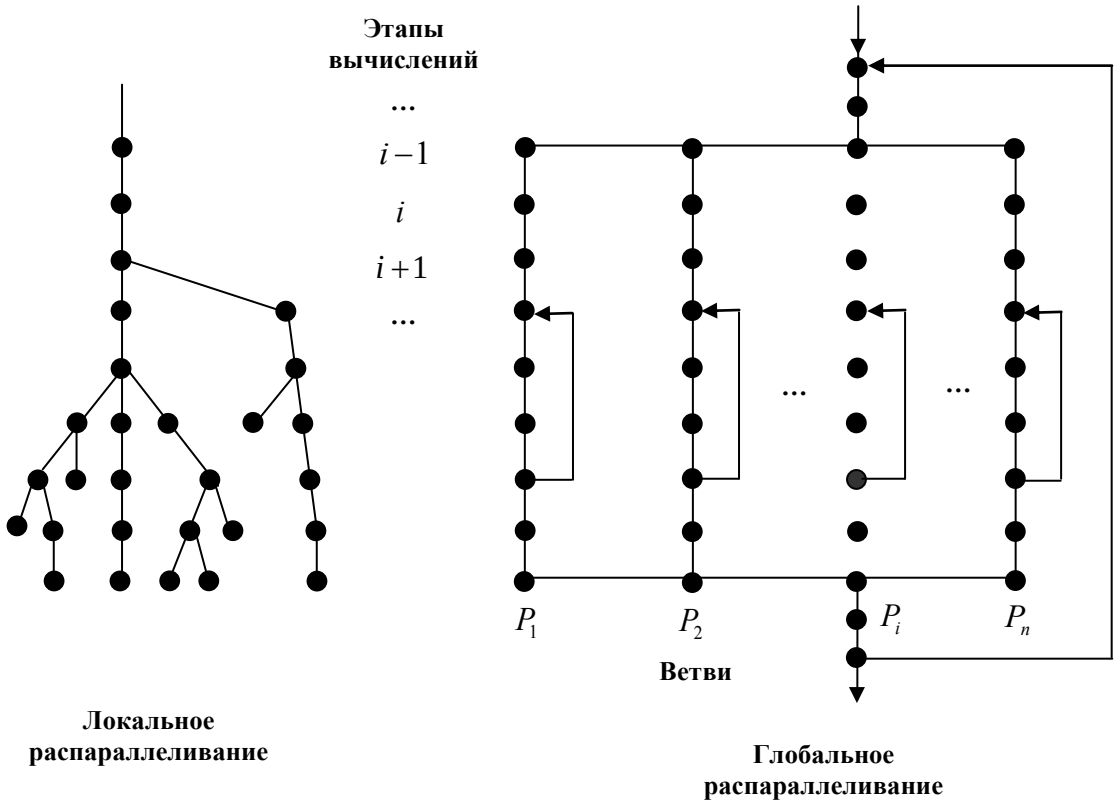

Рис. 3.3. Схемы параллельных алгоритмов

● – оператор

Второй подход ориентирован на разбиение сложной задачи на крупные блоки–подзадачи, между которыми существует слабая связность. Тогда в алгоритмах, построенных на основе крупноблочного распараллеливания, операции обмена между подзадачами будут составлять незначительную часть по сравнению с общим числом операций в каждой подзадаче. Такие подзадачи называют *ветвями параллельного алгоритма*, а соответствующие им программы – *ветвями параллельной программы*.

Пусть для выполнения параллельной программы выделяется (под)множество $C = \{c_i\}, i \in \{1,2,...,n\},$ вычислений c_i, тогда

$$P = \bigcup_{i=1}^{n} P_i$$

и P_i–ветвь, реализуемая вычислителем c_i . Принцип однородности коллектива вычислителей позволяет создавать параллельные алгоритмы и программы с идентичными ветвями.

Одним из конструктивных приемов крупноблочного распараллеливания сложных задач является *распараллеливание по циклам*. Прием позволяет представить процесс решения задачи в виде параллельных ветвей, полученных расщеплением цикла на части, число которых в пределе равно числу повторений цикла. На входе и выходе из цикла процесс вычислений – последовательный (рис.3.3); доля последовательных участков в общем времени решения задачи незначительна. Говоря языком физика, процесс вычислений можно представить в виде “пучностей” – параллельных ветвей, периодически перемежающихся “узлами” – последовательными процессами. На переходах между

пучностями и узлами выполняются функции управления параллельными вычислительным процессом и обмены информацией между ветвями.

Как правило, последовательные участки имеют место в начале и конце параллельной программы. На начальном участке осуществляется инициирование программы и ввод исходных данных, а на конечном – вывод результатов (запись в архивные файлы). При достаточно полном воплощении принципов модели коллектива вычислителей допустимы реализации в системе параллельных ввода и вывода информации.

Последовательные участки в параллельной программе также могут иметь место, если используются негрупповые обмены информацией между ветвями (вычислителями).

Параллельная программа решения сложной задачи допускает представление в виде композиции

$$P_1 * P_2 * \ldots * P_i * \ldots * P_n,$$

в которой P_i есть i-я ветвь программы, n–допустимое число ветвей. Подчеркнем: каждая из ветвей P_i реализуется только на одном вычислителе коллектива, а информационные связи между ее операторами и операторами других ветвей $\{P_j\}, \quad j \neq i,$ осуществляются при помощи специальных операторов обмена информацией.

В дальнейшем, для краткости, наряду с терминами “параллельный алгоритм” и “параллельная программа” будем использовать соответственно “P-*алгоритм*” и “P-*программа*”.

3.3.2. Параллельный алгоритм умножения матриц

Структуры параллельных алгоритмов (и, следовательно, P-программ) определяются графами информационных и управляющих связей, вершинам которых сопоставлены операторы ветвей, а ребрам информационные и управляющие связи между операторами. Анализ прямых и итерационных методов вычислительной математики показывает, что в их основе лежат, как правило, операции над матрицами и векторами данных.

Проиллюстрируем методику крупноблочного распараллеливания на примере умножения матриц больших размеров. Пусть требуется построить параллельный алгоритм, вычисляющий произведение двух прямоугольных матриц:

$$\boldsymbol{A}[1:N,1:K] \times \boldsymbol{B}[1:K,1:M] = \boldsymbol{C}[1:N,1:M]$$

или, что то же самое,

$$\begin{Vmatrix} a_{11} & a_{12} & \cdots & a_{1h} & \cdots & a_{1K} \\ a_{21} & a_{22} & \cdots & a_{2h} & \cdots & a_{2K} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i1} & a_{i2} & \cdots & a_{ih} & \cdots & a_{iK} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{N1} & a_{N2} & \cdots & a_{Nh} & \cdots & a_{NK} \end{Vmatrix} \times \begin{Vmatrix} b_{11} & b_{12} & \cdots & b_{1j} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2j} & \cdots & b_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{h1} & b_{h2} & \cdots & b_{hj} & \cdots & b_{hM} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b_{K1} & b_{K2} & \cdots & b_{Kj} & \cdots & b_{KM} \end{Vmatrix} =$$

$$= \begin{Vmatrix} c_{11} & c_{12} & \cdots & c_{1j} & \cdots & c_{1M} \\ c_{21} & c_{22} & \cdots & c_{2j} & \cdots & c_{2M} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{i1} & c_{i2} & \cdots & c_{ij} & \cdots & c_{iM} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ c_{N1} & c_{N2} & \cdots & c_{Nj} & \cdots & c_{NM} \end{Vmatrix},$$

где элементы матрицы-произведения $C[1:N,\ \ 1:M]$ вычисляются по формуле

$$c_{ij} = \sum_{h=1}^{K} a_{ih} b_{hj}, \quad i = \overline{1,N}, \quad j = \overline{1,M}. \tag{3.8}$$

Допустим также, что параллельный алгоритм ориентирован на реализацию в вычислительной системе, состоящей из n вычислителей.

Пусть размеры $N \times K$ и $K \times M$ матриц $\boldsymbol{A}$ и $\boldsymbol{B}$ достаточно большие и таковы, что имеют место неравенства $N >> n, \quad K >> n, \quad M >> n$. При параллельной обработке необходимо, чтобы каждый вычислитель производил расчет своих элементов матрицы $\boldsymbol{C}$. При этом легко заметить, что размещение матриц $\boldsymbol{A}$ и $\boldsymbol{B}$ целиком в каждом вычислителе требует большой суммарной емкости памяти. Минимум емкости памяти будет достигнут, если каждая из исходных матриц будет разбита на n равных частей, и в каждый вычислитель будет размещено по одной такой части матриц $\boldsymbol{A}$ и $\boldsymbol{B}$. Например, каждую из матриц $\boldsymbol{A}$ и $\boldsymbol{B}$ можно разрезать на n равных соответственно горизонтальных и вертикальных полос. Причем в первом вычислителе можно разместить строки $1,2,...,\ ]N/n[$ и столбцы $1,2,...,\ ]M/n[$; в l-ом вычислителе – строки $(l-1)\cdot]N/n[+1,\ (l-1)\cdot]N/n[+2,...,\ l\ ]N/n[$ и столбцы $(l-1)\cdot]M/n[+1,\ (l-1)\cdot]M/n[+2,...,\ l\ ]M/n[$; в n-ом вычислителе – строки $(n-1)\cdot]N/n[+1,\ (n-1)\cdot]N/n[+2,...,\ n\ ]N/n[$ и столбцы $(n-1)\cdot]M/n[+1,\ (n-1)\cdot]M/n[+2,...,\ n\ ]M/n[$ матриц $\boldsymbol{A}$ и $\boldsymbol{B}$, соответственно. Через $]x[$ обозначено такое ближайшее к x целое число, для которого справедливо неравенство $]x[\geq x$. В результате будет получено однородное распределение данных по вычислителям коллектива (рис.3.4).

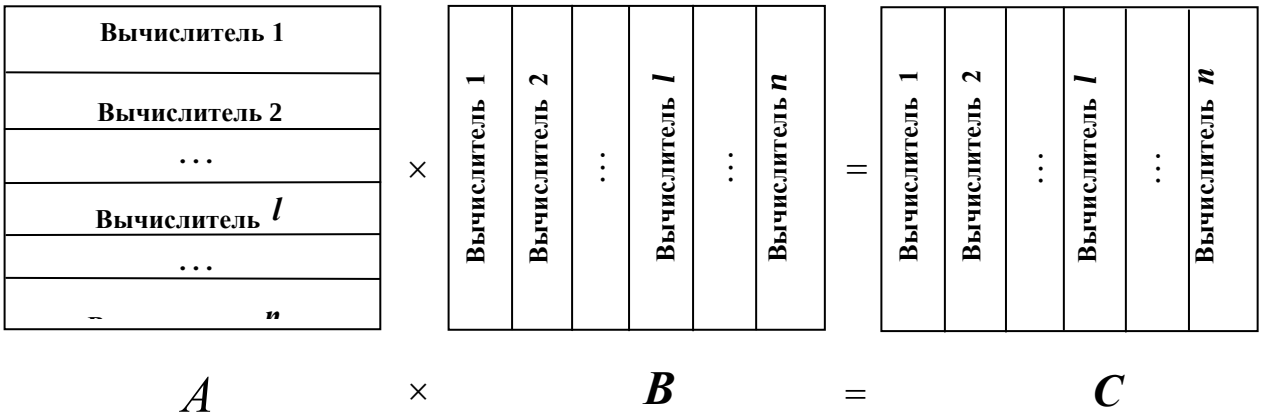

Рис.3.4. Распределение данных по вычислителям ВС

Параллельный вычислительный процесс можно организовать следующим способом. Сначала первый вычислитель передает остальным вычислителям первую строку из своей полосы матрицы A. После этого все вычислители, используя формулу (3.8), осуществляют параллельный расчёт $]M/n[$ элементов первой строки матрицы C. Затем первый вычислитель рассылает во все остальные вычислители вторую строку своей полосы матрицы A и производятся вычисления элементов второй строки матрицы C и т.д., до тех пор пока первый вычислитель не перешлет все строки своей части матрицы A. После этого пересылками будут заниматься последовательно второй вычислитель, третий вычислитель и т.д., наконец, n-ый вычислитель. Матрица C получается распределенной по вычислителям, причем в каждом будет своя вертикальная полоса (рис. 3.4).

Итак, вследствие однородного распределения данных получены одинаковые ветви параллельного алгоритма, однако при этом ветви используют различные части данных. Так как для каждой ветви своих данных недостаточно, то ветви (точнее реализующие их вычислители) вступают во взаимодействия (между ними осуществляются обмены информацией). Операторная схема l-й ветви P-алгоритма, реализуемая на вычислителе с номером l, $1 \le l \le n$, представлена на рис.3.5.

Операции над матрицами и векторами (матрицами-строками или матрицами-столбцами) данных широко используются не только в прямых, но и в итерационных методах вычислительной математики. Так, алгоритм решения системы линейных алгебраических уравнений итерационным методом сводится к вычислению

$$x_i^{(k+1)} = b_i + \sum_{j=1}^{L} a_{ij} x_j^{(k)}, \ i = 1,2,...,L,$$

где $x_i^{(k+1)}$ – значение i-й переменной, рассчитываемое в $(k+1)$-й итерации; b_i и a_{ij} – коэффициенты, составляющие матрицу-столбец (вектор-столбец) и матрицу, соответственно;

$$\max \left\{ \left| x_i^{(k+1)} - x_i^{(k)} \right| \right\} < \delta$$

есть условие окончания итераций.

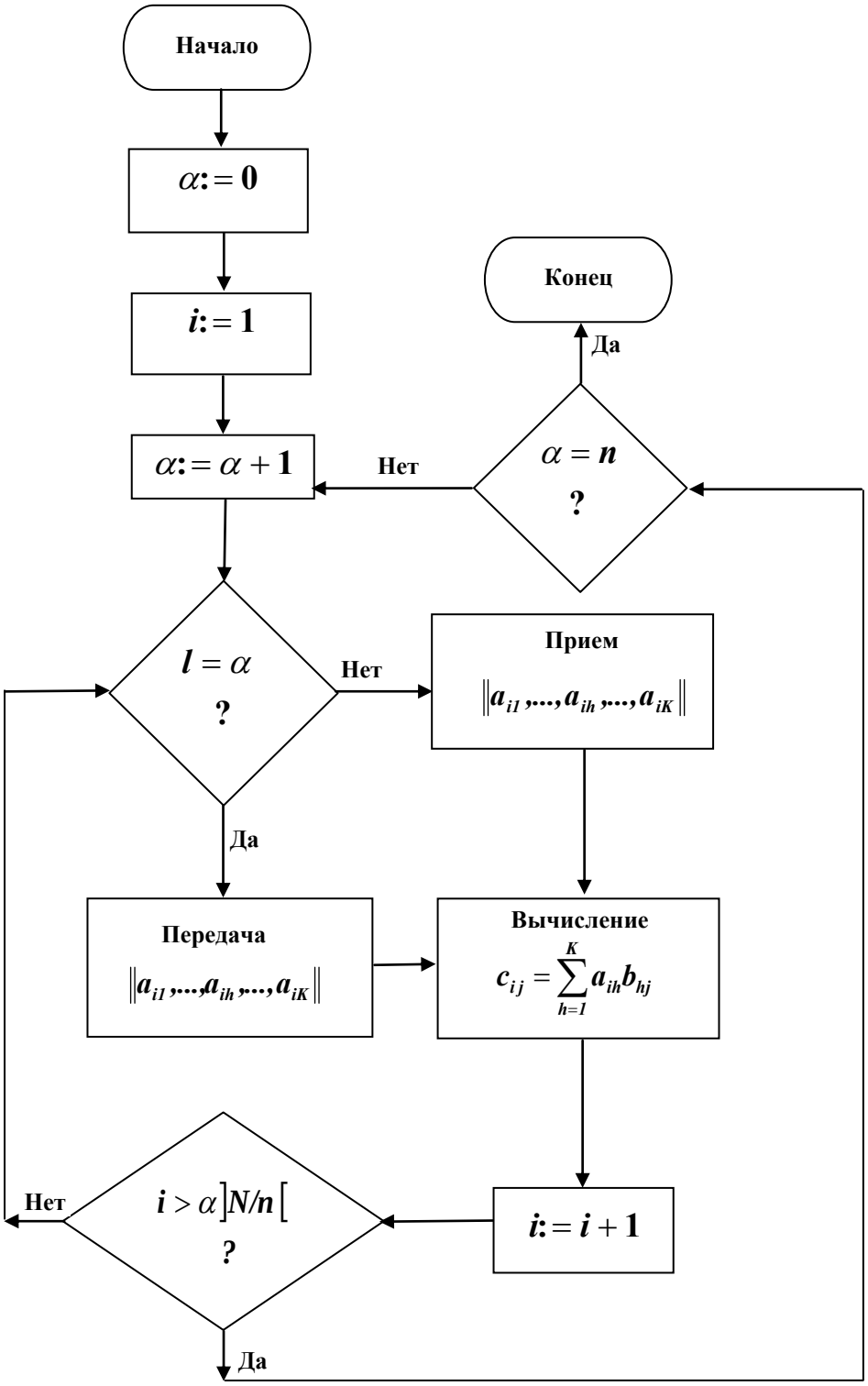

Рис.3.5. Схема ветви параллельного алгоритма умножения матриц

$]M/n[\cdot(l-1) < j \le]M/n[\cdot l$, α – номер передающего вычислителя,

$\{1,2,...,\alpha-1, \alpha+1,...,n\}$ – номера принимающих вычислителей