Многопроцессорные вычислительные системы
2006. №2(12)

УДК: 004.43

© 2006 г. **И.Л. Артемьева**, канд. техн. наук,
М.Б. Тютюнник
(Институт автоматики и процессов управления ДВО РАН, Владивосток)

РАСПАРАЛЛЕЛИВАНИЕ ВЫЧИСЛЕНИЙ ДЛЯ МОДУЛЬНЫХ СИСТЕМ ПРОДУКЦИЙ[1]

Данная работа описывает исследования, целью которых является разработка системы параллельного программирования на основе конфлюэнтных продукций. В ней рассматривается метод организации параллельных вычислений для логических программ, число модулей которых больше одного.

Введение

В работах [1, 2] даны описания схем распараллеливания вычислений для системы конфлюэнтных продукций с использованием множества активных правил. В работе [3] описаны схемы распараллеливания вычислений для правил одного модуля логической программы. В последних схемах для организации параллельных вычислений используется информационный граф. Во всех перечисленных работах содержится также описание результатов исследований рассмотренных схем.

Однако для сложных задач программа на продукционном языке состоит из совокупности модулей, что требует использования других схем управления вычислениями, в которых бы учитывалось наличие множества модулей. Традиционно каждый из модулей описывает правила решения некоторой подзадачи. Целью данной работы является описание схемы распараллеливания вычислений для программ, число модулей которых больше одного.

Постановка задачи

Программа, записанная на продукционном языке, состоит из совокупности модулей, каждый из которых содержит правила решения одной

[1] Работа выполнена при финансовой поддержке ДВО РАН в рамках программы № 14 Президиума РАН, проект 06-I-П14-052.

из подзадач решаемой программой задачи. Один модуль содержит описание переменных, значения которых задают входные и выходные данные модуля, продукционные правила, описывающие метод решения подзадачи.

В логической программе модуль вызывается следующим образом: ИмяМодуля(НаборИменПередаваемыхОбъектов, НаборИменПолучаемыхОбъектов), при этом вызов модуля происходит, если выполняется условие правила активации модуля (условие некоторого правила). Здесь ИмяМодуля – идентификатор вызываемого модуля. Модуль с таким именем должен быть описан заранее. НаборИменПередаваемыхОбъектов – это множество имен объектов, значения которых передаются в вызываемый модуль; НаборИменПолучаемыхОбъектов – множество имен объектов, значения которых принимаются из вызванного модуля и синхронизируются с имеющимися значениями.

Каждый модуль состоит из нескольких секций, в которых описываются данные и правила. Параметрами функции вызова модуля могут быть только те имена объектов, которые входят в секцию описания глобальных объектов вызываемого модуля.

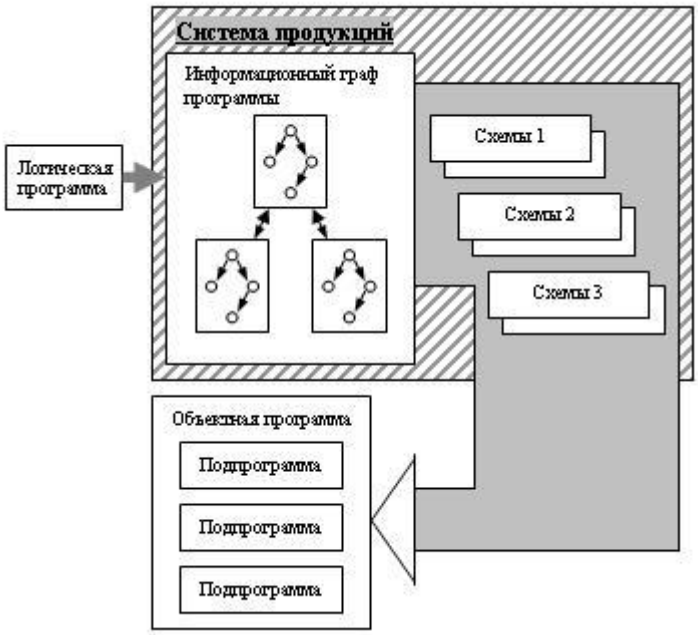

Рис. 1. Работа языкового процессора для системы конфлюентных продукций.

Языковой процессор (рис. 1) представляет собой компилятор, преобразующий текст на продукционном языке в объектную программу на алгоритмическом языке высокого уровня [3]. Объектная программа реализует процесс логического вывода, задаваемый программой.

До генерации объектной программы языковой процессор строит информационный граф программы и анализирует его свойства. Понятие информационного графа для программы с одним модулем было дано в работе [3]. Обобщим данное понятие на случай программы, число модулей которой больше одного.

Информационным графом (ИГ) программы будем называть ориентированный цикличный граф, вершинами которого являются информационные графы [3] модулей программы, а дуги обозначают информационные связи между графами модулей. Таким образом, дуга соединяет два графа модулей, если в правиле одного модуля описан вызов другого модуля.

ИГ программы может состоять из множества ветвей, которые система должна выделить и по возможности начать обрабатывать, используя доступные ресурсы параллельной ЭВМ. Неизвестным является время обработки модулей и правил в модулях. Поэтому главной целью является минимизация всего времени выполнения логической программы посредством распределения заданий по вычислительным узлам и оптимизации вычислений с помощью соответствующих схем распараллеливания.

Параллельные вычисления, состоящие в выполнении набора параллельных зависимых по данным заданий на многопроцессорной ЭВМ, можно представить в виде пары $C = \langle P, ПР \rangle$, где $P = \{p_1, \ldots, p_p\}$ – множество процессов, доступных для параллельных вычислений; ПР – логическая программа. Проблема состоит в том, чтобы таким образом назначить на обработку правила, описанные в ПР, процессам P, чтобы время выполнения было минимальным.

Заметим, что система, выполняющая модульную программу, должна знать до начала исполнения, какое количество процессов кластера доступно для вычислений. Количество процессов, как и количество процессоров (узлов компьютера), определяется сторонними средствами до запуска системы продукций и зависит от конфигурации конкретной ЭВМ.

Определение информационного графа модульной программы

Введем обозначения.

$ПР = \langle M \rangle$, $M = \{m_1, \ldots, m_m\}$, где ПР – логическая программа; M – множество логических модулей; $m_i = \langle П \rangle$, $П = \{\pi_1, \ldots, \pi_\pi\}$, где П – множество правил модуля m_i.

GПР = (GM, MD) – ориентированный цикличный граф программы (ИГ программы), где M – множество вершин графа, причем каждая вершина соответствует графу логического модуля m программы; $MD = \{(GM_i, GM_j): \exists! \pi_i \in GM_i: m_j \in THEN(\pi_i), i \neq j\}$ – множество дуг графа.

GM = (П, ED) – ориентированный цикличный граф логического модуля m, где П – множество вершин графа, причем каждая вершина соответствует правилу π программы; $ED = \{(\pi_i, \pi_j): THEN(\pi_i) \cap IF(\pi_j) \neq \varnothing, i \neq j\}$ – множество дуг графа.

$Mark(GM) = \{(mark(m_i))\}$ – множество меток для вершин графа GM,

где $mark(m_i) = \begin{cases} 0, \text{если предок}(m_l) = \varnothing \\ 1 + \max\limits_{m_k \in предок(m_i)} \{ mark(m_k) \}, иначе \end{cases}$.

Пар1(m_i) = {m_i, m_{i1}, …, m_{iz}}: mark(m_i) = mark(m_{i1}) = mark(m_{i2}) = … = mark(m_{iz}) – множество модулей, которые можно начать выполнять параллельно с модулем m_i и друг с другом.

ПарМ(m_i) = {m_i, m_{i1}, …, m_{iz}}: ∀mark’∈{mark(m_{i1}), …, mark(m_{iz})) mark(m_i) ≤ mark’ < max(mark(потомок(m_i))) – множество модулей, которые можно выполнять параллельно с модулем m_i (но не обязательно параллельно друг с другом).

Множество ПарМ(m_i) состоит из объединения Пар1(m_j) для всех m_j∈ПарМ(m_i). При этом максимально возможное количество процессов, работающих параллельно с модулем m_i, равно максимальному количеству элементов в любом из множеств Пар1(m_j) плюс один процесс, выделяемый для m_i. Следовательно, число требуемых программе процессов не превышает максимального количества элементов из ПарМ(m_i) для всех правил, входящих в программу. Это условие действует лишь в том случае, когда процесс обрабатывает целое правило (а не кортежи по отдельности, например).

Из введенных обозначений следует, что имеется двухуровневый граф, верхний уровень которого представляет собой граф программы, состоящей из модулей, а нижний уровень описывает графы каждого модуля, состоящего из правил.

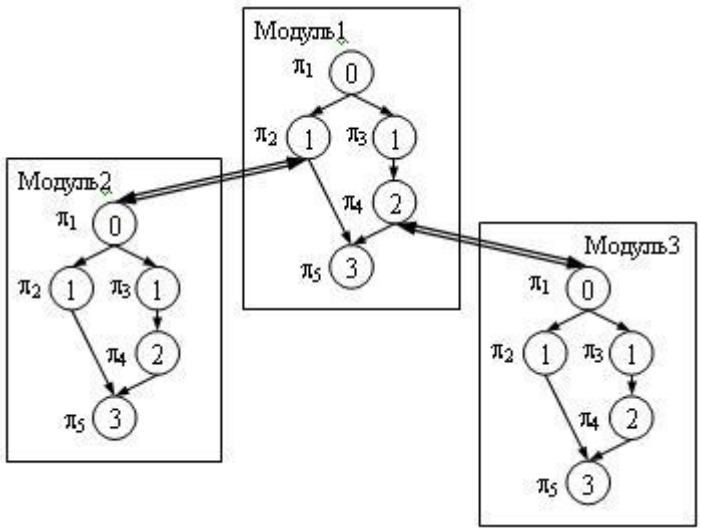

Рис. 2. Пример информационного графа программы.

На рис. 2 приведен пример ИГ логической программы. Программа состоит из трех модулей, каждый из которых описывает некоторый набор правил; для каждой вершины ИГ определена своя метка. Двойные стрелки обозначают вызов модуля и передачу данных в вызываемый модуль и обратно.

Правила π_2, π_4 модуля 1 содержат вызовы модулей 2 и 3 соответственно. Так как ПарМ(π_2) = {π_3, π_4}, то модули 2 и 3 могут исполняться в параллельном режиме.

Следует заметить, что программы, структура которых аналогична

структуре программы, представленной на рис. 2, являются достаточно распространенными, так как практика показывает, что реальная задача обычно содержит не более десятка модулей. Однако многопроцессорная ЭВМ предоставляет ограниченное количество узлов и, соответственно, процессов, которые можно передать системе. Поэтому необходимо задать механизм, позволяющий применить схему распараллеливания в зависимости от структуры логической программы и ограничений, налагаемых ЭВМ.

Ограничения могут быть следующими:

1. Количество доступных системных процессов μ(P). Если μ(P) больше или равно количеству правил в модулях программы, то данный случай удобен для вычислений, так как можно заранее каждому процессу назначить на обработку конкретное правило. Если μ(P) меньше, то необходимо использовать схемы по поочередному назначению правил процессам, рассматривая при этом каждый модуль отдельно.

2. Время выполнения отдельного модуля и правил. При вычислении отдельного модуля может возникнуть ситуация, когда этот модуль будет обрабатываться во много раз дольше, чем любой другой логический модуль. В этом случае может возникнуть длительный простой системы в ожидании завершения вычислений данного модуля. Одним из выходов является использование других схем внутри модуля для более эффективного исполнения правил.

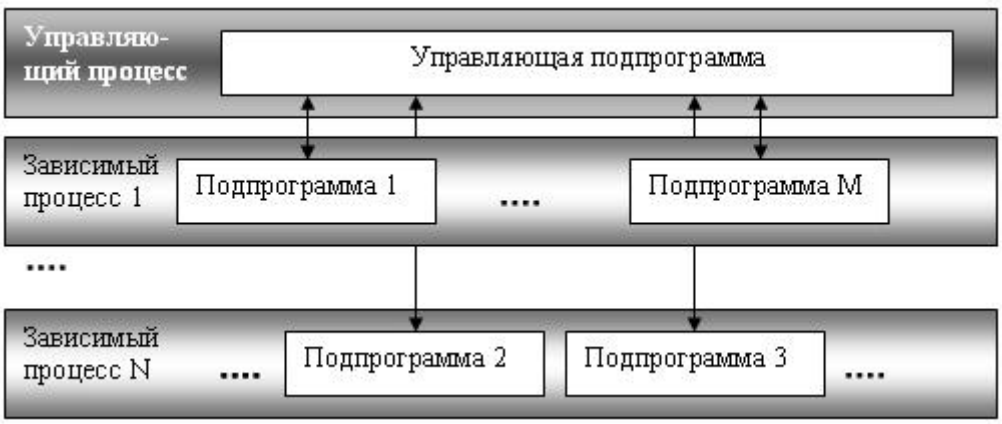

Рис. 3. Схема взаимодействия процессов.

В работах [1, 3] были описаны схемы распараллеливания процесса логического вывода для правил внутри отдельного модуля. Метод распараллеливания предлагал использовать архитектуру типа «клиент-сервер» (рис. 3), когда отдельному процессу предлагается роль диспетчера (управляющий процесс), остальным процессам – роль обрабатывающего процесса (зависимые процессы). Управляющий процесс производит ввод-вывод данных и их синхронизацию, а также обмен данными с зависимыми процессами; он производит старт подпрограмм в зависимых процессах и определяет очередность выполнения правил. Каждый зависимый процесс выполняет подпрограмму, реализующую процесс логического вывода, т.е.

вычисляет правило (либо часть правила, либо группу правил), а результирующие данные передает в управляющий процесс.

Схемы распараллеливания внутри модуля использовали все имеющиеся процессы для вычислений правил. Однако при выполнении логической программы, содержащей набор модулей, можно столкнуться с ситуацией, когда количество модулей будет больше, чем доступное количество процессов. В данном случае необходим механизм распределения процессов по модулям и задания очередности выполнения каждого модуля.

Распределение модулей по процессам

Для управления модулями выделяем отдельный процесс (процесс управления модулями), который будет производить ввод-вывод, анализ и синхронизацию данных, вызов управляющих подпрограмм для правил и назначение процессов (рис. 4).

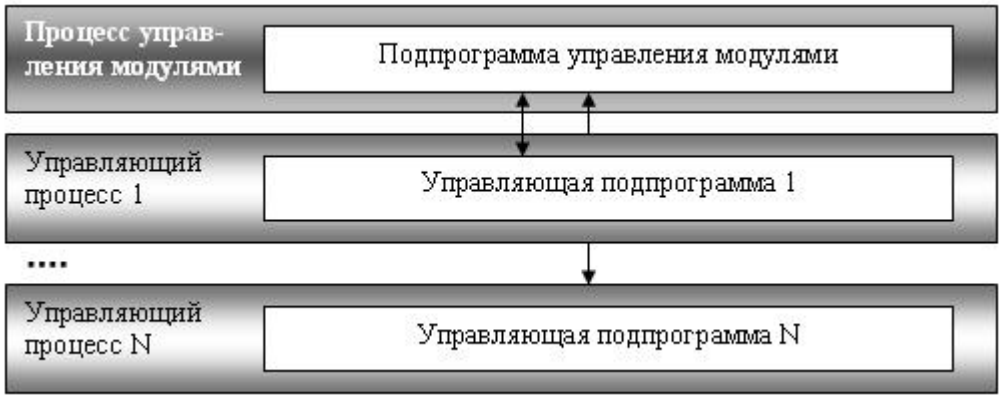

Рис. 4. Схема взаимодействия управляющих процессов.

Очередность выполнения модулей определяется с помощью информационного графа программы. Анализ графа показывает количество ветвей графа, которые можно обрабатывать параллельно, т.е. заранее известно число процессов, необходимых для выполнения всех модулей в параллельном режиме. Используя информационные графы каждого модуля, можно определить количество процессов, необходимых для вычисления правил внутри модулей.

При вызове модуля процесс управления модулями назначает вызываемому модулю свободные процессы, число которых равно необходимому количеству процессов для этого модуля. Если модуль, где произошел вызов, не может передать достаточное количество процессов, он передает процессы по мере того, как они освобождаются, закончив вычисления правил. Если в модуле не вычисляется ни одно правило, кроме того, в котором произошел вызов модуля, то процесс управления модулями может передать все свободные процессы в вызванный модуль.

Если логические модули не содержат циклических вызовов, то информационный граф не будет иметь циклов, а значит, на этапе трансляции

программы можно определить такой порядок выполнения модулей, при котором модули будут выполняться не более одного раза, т.е. последовательность выполнения детерминирована. Для этого на этапе трансляции строятся структуры данных, которые описывают как информационный граф, так и очередность выполнения модулей на этапе исполнения.

Если логические модули содержат циклические вызовы, то информационный граф будет иметь циклы. В этом случае последовательность выполнения модулей представляет собой детерминированную последовательность циклов или одиночных модулей. Каждый цикл объединяет совокупность модулей, которые следует выполнять в соответствии с очередностью, определенной алгоритмом на этапе трансляции. Каждый цикл выполняется до тех пор, пока модули из цикла изменяют состояние вывода. В случае, когда в цикл вложен еще один цикл, происходит раскрытие цикла, т.е. цикл представляется в виде отдельного графа, являющегося подграфом логической программы, обработка модулей в нем происходит с теми же условиями, что и при работе с общим графом.

В первую очередь на этапе трансляции строятся вспомогательные структуры данных, которые описывают ИГ программы (GПР-ориентированный цикличный граф программы). Далее определяется множество меток (mark(GПР) – конечное (упорядоченное) множество меток), с помощью которых вычисляется число модулей, которые можно выполнять параллельно. Множество меток определяет последовательность выполнения модулей.

Теперь система обладает всеми данными для того, чтобы начать исполнять программу. Механизм управления в главном процессе-диспетчере должен сначала проинициализировать все входные данные и запустить процессы для вычисления модулей. Для примера на рис. 2 очередность выполнения модулей будет следующей.

№	1	2	3
№ метки	0	1	1
№ модуля	1	2	3

В приведенной таблице показано, что стоящие справа в ячейках модули не могут начать работу раньше, чем начнут выполняться стоящие слева модули.

В течение работы каждый процесс выполняет подпрограммы, соответствующие отдельному модулю и построенные на этапе трансляции. Так как процессы работают каждый со своим участком памяти (в виду специфики кластера), им необходимо пересылать друг другу результирующие данные и данные синхронизации.

Каждый процесс производит одинаковые действия по обработке модулей и правил. Это получение необходимых данных для каждого модуля от управляющего процесса, синхронизация полученных данных с уже

имеющимися в наличии, вычисление правил и отправка результата работы зависимым модулям и процессам. Роль управляющего процесса сводится к начальной инициализации данных, запуску всех рабочих процессов и получению необходимых результатов вычисления правил.

Введем следующие обозначения: Pp – количество всех доступных процессов; Pt – количество свободных процессов в текущий момент; Pw_i – количество выделенных для работы процессов для модуля m_i; $Max(ПарМ(m_i)) \equiv Pmax_i$, ($Pmax_i \geq 2$) – оптимальное количество процессов, необходимых для вычислений модуля m_i с применением схемы, использующей информационный граф GM_i модуля m_i; $Pmax_i$ должно быть равно минимум двум, так как один процесс занимается управляющей подпрограммой, второй выделяется для обработки правил; Предок(m) – модуль, вызвавший модуль m. Если модуль является начальным (корневым на графе GПР), то Предок(m) = ∅.

Таким образом, оптимальное число процессов для выполнения программы задается схемой распараллеливания, использующей информационный граф GM для каждого модуля. Это число определяет количество процессов, которое можно выделить для того, чтобы можно было выполнять максимальное число правил в параллельном режиме. Минимальным количеством процессов, выделяемых для модуля, являются два процесса: процесс-диспетчер и зависимый процесс. Отсюда следует, что на всю программу должно быть выделено число процессов, равное минимум двойному количеству всех модулей плюс один (μ(M)*2 + 1). При этом количество свободных процессов будет уменьшаться постепенно, по мере вызова модулей.

Для примера приведем алгоритм работы модулей.

1. Начало.
2. Выделяем для вычислений корневого модуля m_0 все свободные процессы из Pp. m = m_0.
3. Вычисляем модуль m.
4. Если встретили вызов модуля m_i, то выделяем Pw_i ($Pw_i = \min(Pmax_i, Pt)$) процессов для вычислений модуля m_i.
    a. Если $Pw_i < 2$, то вычисляем m до тех пор, пока не появится столько освободившихся процессов P, что $Pw_i + P = 2$.
    b. Если $Pw_i \geq 2$, то m = m_i, пункт 3.
    c. Если $2 \leq Pw_i < Pmax_i$, то каждый освободившийся процесс передаем для вычисления mi до тех пор, пока Pw_i не будет равно $Pmax_i$.
5. После вычисления модуля m все результирующие данные передаем в Предок(m).
    a. Если Предок(m) = ∅, то пункт 6.
    b. Если Предок(m) ≠ ∅, то принимаем результирующие данные из m в Предок(m), синхронизируем их. Каждый освободившийся процесс передаем для вычисления Предок(m) до тех пор, пока $Pw_{Предок(m)}$ не будет равно $Pmax_{Предок(m)}$. m = Предок(m), пункт 3.
6. Конец.

Заключение

В работе рассмотрена схема распараллеливания для системы параллельного программирования, входным языком которой является модульный продукционный язык. Описанная схема распараллеливания относится к классу глобальных схем, т.е. применяется к модулям логической программы.

Схема, описанная в работе, использует информационный граф модульной программы, построенный на основе связей между логическими модулями программы. Каждому модулю назначается набор процессов, которые осуществляют вычисление правил и передачу результирующих данных. Такая схема позволяет построить эффективный алгоритм выполнения модульной системы продукций для многопроцессорной ЭВМ.

ЛИТЕРАТУРА

1. *Артемьева И.Л., Тютюнник М.Б.* Схемы распараллеливания вычислений для системы конфлюэнтных продукций // Информатика и системы управления. – 2005. – № 2(8). – С.102-112.
2. *Артемьева И.Л., Тютюнник М.Б.* Методы распараллеливания вычислений для системы параллельного программирования на основе декларативных продукций // Тр. II междунар. конф. "Параллельные вычисления и задачи управления". – М.: ИПУ РАН, 2004. – С.727-737.
3. *Тютюнник М.Б.* Использование информационного графа при распараллеливании вычислений для системы конфлюэнтных продукций // Информатика и системы управления. – 2006. – № 1(11). – С.181-192.

Статья представлена к публикации членом редколлегии А.С. Клещевым.