Сравнительная оценка способов реализации распределенного дерева логического вывода

Пушкаренко С.А., Дацун Н.Н.
Донецкий национальный технический университет
S.Pushkarenko@it.stirol.net

Abstract

Pushkarenko S., Datsun N. "Comparative estimation of methods of realization of distributed deduction tree". *The article examines methods of paralleling of Datalog program translation process. There were reviewed possible alternatives of realization of distributed deduction tree in deductive databases.*

Введение

Современные базы данных (БД) позволяют хранить огромные объёмы информации. Однако использование этой информации не достаточно эффективно. Применяемый в настоящее время язык запросов SQL не позволяет в полной мере осуществить извлечение всей необходимой информации из БД. Многие запросы не могут быть представлены одним SQL–оператором. Такие запросы теряют свою декларативность, становясь процедурными, сложными для написания и восприятия, что повышает вероятность ошибок. В связи с этим возникает необходимость в альтернативных средствах извлечения информации. Наиболее перспективным является использование в качестве языков запросов к реляционным БД языков логического программирования, таких как Пролог и Дейталог. Такие системы называют дедуктивными базами данных (ДБД).

В статье рассматриваются способы распараллеливания процесса трансляции Дейталог-программы, а также проводится сравнительная оценка возможных вариантов реализации распределённого дерева логического вывода в дедуктивных базах данных (ДБД) на многопроцессорных вычислительных системах (МВС) с распределённой памятью. В качестве средств моделирования использованы разработанный авторами транслятор языка Дейталог [1], среда Visual C++ и библиотека MPI.

Современные подходы к реализации параллелизма в логической части Дейталог-программ

Можно выделить такие виды параллелизма в логической части Дейталог-программ:

– «И»-параллелизм [2];
– «ИЛИ»-параллелизм [2];
– мультитрединг.

«И»-параллелизм имеет место в таких случаях: когда подцели, принадлежащие данному запросу, выполняются параллельно различными вычислительными модулями, либо параллельно конкретизируются аргументы отдельной подцели запроса;

«ИЛИ»-параллелизм имеет место, когда одна подцель может быть унифицирована с заглавиями нескольких клауз и тела этих клауз вычисляются различными вычислительными модулями.

Мультитрединг есть получение и обработка одновременно «многих» потоков команд и данных: при этом устройство управления распределяет потоки для параллельного выполнения на внутренних вычислительных устройствах процессора.

В современных трансляторах языков логического программирования реализованы различные варианты параллелизма:

– «И»-параллелизм (транслятор Parlog, ф.Parallel Logic Programming Ltd [3]); ;
– «ИЛИ»-параллелизм (Акторный Пролог, автор Морозов А. А. [4]; YAP Prolog, LIACC/ Universidade do Porto [5]; BinProlog, ф. BinNet Corp. [6]);
– мультитрединг (BinProlog, ф. BinNet Corp. [6]; QuProlog, University of Queensland [7]; SWI Prolog, The SWI-Prolog Foundation [8]).

Достижение большего быстродействия вычислительной системы (ВС) возможно двумя методами: экстенсивным (улучшение архитектуры и процессоров, увеличение тактовой частоты) или интенсивным (оптимизация, распараллеливание).

Экстенсивный метод может быть реализован путем увеличения быстродействия и продуктивности ВС за счёт организации и расширения кластеров, сложных многопроцессорных систем.

Интенсивный метод может быть реализован путем перенесения части функций по

распараллеливанию вычислительного процесса программы на фазу трансляции.

Перспективным представляется сочетание экстенсивного и интенсивного методов.

В работе [9] выполнен сравнительный анализ реализации логической части Дейталог-программы на последовательной и параллельной архитектурах. Предложен подход к организации внутреннего представления экстенсиональной части программы во внутренней памяти, выполненный при трансляции программы. Этот подход позволяет повысить эффективность выполнения запроса в классической и параллельной реализациях. Для моделирования параллелизма с помощью MPICH использована модель гомогенного симметричного кластера. Исследование модели показало, что параллельное выполнение на n процессорах (n=3) начинает давать прирост производительности при фиксированном количестве фактов (m) и количестве аргументов запросов k>=10.

В данной работе рассматриваются вопросы реализации параллелизма интенсиональной части Дейталог-программы на различных фазах трансляции программы.

Распараллеливание процесса трансляции

В процессе трансляции Дейталог-программы можно распараллеливать фазу лексического анализа, синтаксического анализа, а также разработать внутреннее представление логической программы для последующего осуществления параллельного выполнения программы (логического вывода).

Время, затрачиваемое транслятором на выполнение фазы лексического анализа ($t_{ЛА}$) складывается из следующих составляющих:

t_{lex} – время разбора входного потока символов (выделение лексемы);

t_{find} – время поиска лексемы в таблице символов;

t_{add} – время, необходимое для занесения лексемы в таблицу символов;

t_{res} – время, необходимое для занесения кода лексемы в выходной поток.

Время разбора входного потока зависит от длины входного потока (программы). Время поиска лексемы в таблице символов при условии реализации таблицы символов в виде хеш-таблицы есть величина постоянная. Время занесения лексемы в эту же таблицу символов также является постоянной величиной.

Таким образом, время, затрачиваемое транслятором на выполнение лексического анализа, зависит только от длины входного потока (размера исходной программы):

$$t_{ЛА} = f(n) \qquad (1)$$

где n – длина входного потока символов.

Распараллеливание выполнения фазы лексического и синтаксического анализа может быть целесообразно в том случае, когда выполняется следующее неравенство:

$$t_1 + t_2 < t_3 \qquad (2)$$

где t_1 - время передачи данных;

t_2 - время получения результата;

t_3 - время выполнения задачи процессором.

На МВС с распределенной памятью передача данных между процессорами занимает значительное время. При этом должна быть также предусмотрен механизм реализации распределенной таблицы символов. Фазы лексического и синтаксического анализа - слишком "мелкие" задачи транслятора и их распараллеливание на МВС с распределенной памятью не эффективно.

Наиболее эффективным является распараллеливание процесса логического вывода. Для этого во время трансляции должно быть сформировано внутреннее представление логической программы, пригодное для параллельного логического вывода.

Выбор структур данных

Наиболее удобной структурой данных для внутреннего представления логической программы является дерево. Дерево вывода представляет собой сильноветвящееся дерево. Каждый узел дерева может ссылаться на произвольное заранее неопределённое количество дочерних узлов [10].

В дальнейшем под термином “дерево” будем понимать именно такое сильноветвящееся дерево.

Узел первого уровня дерева соответствует предикату. Дочерние узлы соответствуют подцелям. Порядок узлов в пределах уровня значения не имеет, т.к. используемый в исследовании язык Дейталог не чувствителен ни к порядку правил в программе, ни к порядку подцелей в теле правила [11].

Опишем выбранную структуру дерева на языке Си:

```
typedef struct tree
{
   int predicate;   /*номер предиката*/
   struct list *var;
   /*список аргументов предиката*/
   struct tree *brotherPtr;
   /*ссылка на следующий узел текущего уровня*/
   struct tree *childPtr;
   /*ссылка на дочерний узел*/
   struct tree *parentPtr;
   /*ссылка на родительский узел*/
}
TREE;
```

Проблемы логического вывода

Процедура логического вывода, используемая в ДБД, является достаточно трудоёмкой задачей. Основной проблемой логического вывода является экспоненциальный рост пространства поиска [12].

Наиболее эффективным способом повышения производительности ДБД в настоящее время является распараллеливание процесса логического вывода [12].

При этом различают распределенный поиск и мультипоиск. В основе распределенного поиска лежит принцип разделения пространства поиска на ряд подпространств, т.е. каждому параллельному процессу выдается своя порция пространства поиска [12].

Варианты реализации распределённого дерева логического вывода

Под термином "распределённое дерево" будем иметь в виду совокупность деревьев на различных узлах МВС, соответствующих единой логической программе. На всех экземплярах такого дерева используется одинаковая стратегия логического вывода [12].

Ниже перечислены возможные варианты передачи дерева процессам:

– передавать последовательно каждый узел в интерактивном режиме во время обхода исходного дерева на сервере;

– "запаковать" (сериализовать) всё дерево целиком (например, с помощью XML) и передать сразу все дерево.

Возможны следующие варианты реализации распределённого дерева логического вывода (РД ЛВ):

– вариант 1 - всем процессам передавать исходное дерево;

– вариант 2 - каждому процессу передавать поддерево (часть исходного дерева, сформированную сервером0.

Алгоритмы реализации распределённого дерева логического вывода

Ниже представлен укрупненный алгоритм первого варианта реализации распределённого дерева. В этом случае целесообразно ввести в структуру TREE поле *id* для идентификации узлов дерева и во время формирования дерева (в период трансляции) пронумеровать каждый узел для возможности однозначной идентификации узлов дерева.

Шаг 1: Всем процессорам во время трансляции передаётся исходное дерево. Передача осуществляется с помощью коллективных функций библиотеки MPI.

Шаг 2: После ввода пользователем запроса сервер выбирает наиболее оптимальный алгоритм доказательства и распределяет задания (поддеревья) по процессорам.

Шаг 3: Сервер отправляет каждому процессору соответствующий идентификатор узла, являющегося корнем поддерева, с которым будет работать данный процессор.

Шаг 4: Процессор получает от сервера идентификатор узла и находит его в дереве. Найденный узел является корнем поддерева, с которым будет работать данный процессор.

Ниже представлен укрупненный алгоритм второго варианта реализации распределённого дерева.

Шаг 1: После ввода пользователем запроса сервер выбирает наиболее оптимальный алгоритм доказательства и распределяет задания (поддеревья) по процессорам.

Шаг 2: Сервер отправляет каждому процессору соответствующее поддерево, с которым будет работать данный процессор.

Критерии оптимальности

Критериями оптимальности при параллельном выполнении запроса Дейталог-программы являются:

– время вычислений;

– время передачи данных;

– объём переданных данных.

В первом случае построение распределённого дерева производится один раз (во время трансляции логической программы) и в дальнейшем транслятор будет затрачивать время только на выполнение запроса.

Время построения распределённого дерева ($t_{трансл}$) складывается из следующих составляющих:

- t_0 – анализ логической программы и построение дерева на сервере;
- t_1 – линеаризация дерева;
- t_2 – формирование пакета для передачи;
- t_3 – передача линеаризованного дерева;
- t_4 – "развёртывание" линеаризованного дерева;

Время выполнения запроса для первого способа построения ($t_{запр1}$) складывается из следующих составляющих:

- t_5 – анализ запроса;
- t_6 – распараллеливание запроса сервером;
- t_7 – передача идентификатора узла дерева узлам МВС, участвующим в выполнении запроса;
- t_8 – поиск узла в дереве.

Целевая функция для первого способа построения дерева:

$$t_{трансл} = t_0 + t_1 + t_2 + \sum i\ t_{3i} + t_4 \qquad (3)$$

$$t_{запр1} = f_5(\text{сложность запроса}) + t_6 + t_7 + f_8(id) \qquad (4)$$

Во втором случае время выполнения запроса ($t_{запр2}$) складывается из следующих составляющих:

- t_0 – анализ логической программы и построение дерева на сервере;
- t_5 – анализ запроса;
- t_6 – распараллеливание запроса сервером;
- t_1 – линеаризация дерева;
- t_2 – формирование пакета для передачи;
- t_3 – передача линеаризованного дерева;
- t_4 – "развёртывание" линеаризованного дерева.

Целевая функция для второго способа построения дерева:

$$t_{запр2} = t_0 + t_5 + t_6 + t_1 + \sum i\,(t_{2i} + t_{3i}) + t_4 \qquad (5)$$

В формулах (3) и (5) i есть порядковый номер узла МВС.

В полученных целевых функциях принято, что передача дерева с помощью коллективных функций библиотеки MPI происходит поочерёдно каждому процессу. Этим объясняется наличие в формулах (3) и (5) знака суммирования.

Выводы

На основе анализа современных подходов повышения производительности машин логического вывода предложен подход, сочетающий использование параллельных/ распределенных вычислений и распараллеливания логической части Дейталог-программы при ее трансляции. Проанализированы условия возможности параллельного выполнения фаз лексического и синтаксического анализа. Предложены структура внутреннего представления интенсиональной части Дейталог-программы и варианты реализации распределенного дерева логического вывода. Сформулированы алгоритмы реализации РД ЛВ и критерии оптимальности при параллельном выполнении запроса в Дейталог-программе для предложенных вариантов реализации РД ЛВ.

Целевые функции получены с условием, что передача дерева с помощью коллективных функций библиотеки MPI происходит последовательно каждому процессу. Спецификация MPI не оговаривает метод передачи сообщения коллективными функциями. Способ передачи зависит от конкретной реализации библиотеки MPI [13, 14].

Аппаратные средства МВС позволяют передавать сообщение всем процессам одновременно. В сегменте коммуникационной среды Ethernet все узлы могут "видеть" дейтаграмму, отправленную другим узлом. Но протоколом TCP/IP не предусмотрена множественная адресация, при которой можно было бы указать в качестве получателя несколько узлов.

Поэтому для повышения эффективности передачи данных РД ЛВ в кластере необходим более подходящий протокол, позволяющий передавать сообщение нескольким процессам одновременно.

Литература

1. Дацун Н.Н., Пушкаренко С.А. Параллелизм в дедуктивных базах данных/ Наукові праці Донецького національного технічного університету. Серія "Інформатика, кібернетика і обчислювальна техніка" (ІКОТ-2007). – Вип. 8 (120). – Донецьк: ДонНТУ, 2007. - с. 76-81.
2. Дацун Н.Н. Параллелизм в Datalog базах данных/ Наукові праці Донецького державного технічного університету. Серія "Проблеми моделювання та автоматизації проектування динамічних систем. – Вип. 10. – Донецьк: ДонДТУ, 1999. - с. 71-76.
3. Официальный сайт Parallel Logic Programming Ltd. http://www.parlog.com/en/parlog.html.
4. Морозов А.А. Введение в Акторный Пролог. - Москва: 2002. http://www.cplire.ru/Lab144/start/r_index.html
5. Yet Another Prolog (YAP). Официальный сайт Computer Science Group/LIACC, Universidade do Porto. http:// www.ncc.up.pt/~vsc/Yap.
6. P. Tarau. BinProlog 2006 version 11.x Professional Edition User Guide. http://www.binnetcorp.com/BinProlog.
7. P. Robinson. Qu-Prolog Home Page http://www.itee.uq.edu.au/~pjr/HomePages/QuPrologHome.html.
8. SWI-Prolog. Официальный сайт. http://www.swi-prolog.org.
9. Пушкаренко С.А., Дацун Н.Н. Оценка производительности экстенсиональной части дедуктивных баз данных/ Theoretical and Applied Aspects of Program System Development (TAAPSD'2007). – The Fourh International Conference. – Berdyansk, 4-9 September 2007. Abtracts. – p.170-175.
10. Ахо А.В., Хопкрофт Д., Ульман Д.Д. Структуры данных и алгоритмы.- М.: Издательский дом "Вильямс", 2000. - 384 с.
11. Чери С, Готлоб Г., Танка Л. Логическое программирование и базы данных: Пер. с англ. - М.: Мир, 1992. - 352 с.
12. Вагин В.Н. и др. Достоверный и правдоподобный вывод в интеллектуальных системах/ Под ред. В.Н. Вагина, Д.А. Поспелова. М.: Физматлитгиз, 2004. - 704 с.
13. MPI: A Message-Passing Interface Standart, June 12, 1995.
14. MPI-2: Extensions to the Message-Passing Interface, November 15, 2003.