РАЗБИЕНИЕ ПРОГРАММЫ НА КАДРЫ. УКЛАДКА КАДРА В РЕШЕТКУ ВЫЧИСЛИТЕЛЬНЫХ УЗЛОВ

Г.Ю. Кравченко

Аннотация

Статья посвящена разбиению программы на кадры, а также прямоугольной укладке кадра в решетку. Программная реализация работ реализована в составе Диалогового высокоуровнего оптимизирующего распараллеливателя(ДВОР).

Введение

Разбиение на кадры – это разбиение программы на части, удовлетворяющие определению кадра.

Кадр – это фрагмент программы, который можно эффективно отобразить на архитектуру компьютера.

Для чего необходимо разбивать программу на кадры и эффективно отображать кадры на архитектуру компьютера?

Для повышения производительности вычислительных систем все чаще используются суперкомпьютеры с параллельной организацией вычислений. Происходит активное развитие параллельных архитектур и методов организации параллельных вычислений, так как параллельные вычисления позволили существенно превзойти уровень производительности вычислительных средств, достигнутых за счет увеличения тактовой частоты. Многопроцессорные вычислительные системы со структурно-процедурной организацией вычислений обеспечивают высокую эффективность параллельных вычислений на широком классе задач [2].

Для таких суперкомпьютеров (их еще называют суперкомпьютеры с архитектурой перестраиваемого конвейера) удобно представлять программу в виде последовательности кадров. Поэтому необходимо наиболее оптимальным образом разбивать программу на кадры и отображать кадры на архитектуру компьютера.

Таким образом, разбиение на кадры и оптимальная укладка кадра на архитектуру компьютера позволяют не только эффективно отображать программу на архитектуру компьютера, но еще и выполнять в ней параллельные вычисления.

На сегодня нет архитектурно независимой модели параллельной программы, которая бы годилась для систем из коммерчески доступных микропроцессоров [4]. Подходы к программированию и эффективной загрузке вычислительных систем с миллионами и более процессоров, объединённых накристальными и межкристальными коммуникационными средами, даже не имеют общепризнанных альтернатив. Поэтому необходимо выделить существенные ограничения архитектуры суперкомпьютеров на базе многоядерных мультитредовых кристаллов, чтобы дать разработчикам программного обеспечения модель, выполнение требований которой гарантирует эффективное исполнение параллельных программ. Эта модель также необходима для выбора направления развития компиляторов, автоматически преобразующих программы на традиционных языках программирования в эффективные параллельные программы суперкомпьютеров на базе многоядерных кристаллов [1].

Задача разбиения программы на кадры

Дан фрагмент программы. Необходимо разбить его на кадры, причем так, чтобы все операции кадра можно было бы выполнить с помощью имеющихся ресурсов и без перенастройки коммутатора.

Все вычисления в пределах одного кадра должны выполняться параллельно. А кадры выполняются последовательно. То есть на конвейере сначала выполняются все операции первого кадра, затем все операции второго кадра и так далее.

Ставится задача минимизации количества кадров.

В качестве кадра может быть рассмотрен как отдельный оператор присваивания или последовательность операторов присваивания, так и конвейеризуемый цикл со всеми операторами, входящими в его тело.

В составе ДВОР был разработан модуль разбиения на кадры.

Алгоритм разбиения на кадры работает с графом вычисления программы.

Разработан алгоритм разбиения фрагмента программы на кадры. Алгоритм работает с определенным классом программ и предъявляет следующие требования:

1. В программе могут присутствовать операторы присваивания, одномерные конвейеризуемые циклы, условные операторы.
2. В программе не должно быть вызовов функций.
3. В одном операторе присваивания не должно затрачиваться ресурсов больше допустимого количества ресурсов компьютера.

Также в проекте делались следующие предположения:

1. Все использования в программе – это чтение данных.

2. Количество ресурсов, затрачиваемое на обращение к данным n-мерного массива равно n, умноженному на количество ресурсов, затрачиваемых на обращение к данным одномерного массива.

Схема алгоритма

Пусть входная программа удовлетворяет требованиям, которые были указаны выше. Тогда в один кадр могут попасть:

1. Подряд стоящие операторы присваивания, если на их выполнение хватает ресурсов компьютера.
2. Одномерный конвейеризуемый цикл, если на его выполнение хватает ресурсов компьютера.
3. Условный оператор, если на его выполнение хватает ресурсов компьютера.

Если для выполнения цикла не хватает ресурсов компьютера, то цикл не может быть помещен в один кадр. Тогда заголовок цикла попадает в отдельный кадр, а каждый оператор присваивания, стоящий в цикле, помещается в отдельный кадр.

Если для выполнения условного оператора не хватает ресурсов компьютера, то условный оператор не может быть помещен в один кадр. Тогда заголовок условного оператора попадает в отдельный кадр, а каждый оператор присваивания, стоящий в ветке условного оператора, помещается в отдельный кадр.

Таким образом, алгоритм последовательно проходит по всем операторам программы, подсчитывая количество ресурсов, необходимых на выполнение каждого из них. Затем, если возможно, он пытается поместить в один кадр подряд стоящие операторы присваивания, конвейеризуемый цикл или условный оператор. Поместить оператор в кадр означает пометить этот оператор соответствующим номером кадра.

Прямоугольная укладка кадра в решетку

Решетка (двумерная решетка) – это множество вычислительных узлов суперкомпьютера, соединенных между собой ребрами – каналами пересылки данных. Два узла решетки с координатами (u,v) и (s,t) соответственно соединены ребром, если |u-s|=1 или |v-t|=1. Предполагается, что каждое ребро решетки реализует полнодуплексную пересылку данных.

Каждый узел решетки содержит память, достаточную для хранения всех необходимых данных.

Задача ставится следующим образом. Дан граф вычисления для одного кадра программы и параметры вычислительной системы:

1. Размеры решетки;
2. Коэффициент отношения скорости пересылки в соседний узел решетки к скорости выполнения операции (сколько тактов занимает пересылка по 1 ребру решетки).

На выходе требуется получить укладку кадра в решетку:

1. Инъективное отображение операций кадра в узлы решетки.
2. Для каждой дуги графа — маршрут передачи данных на решетке. Маршруты могут пересекаться.
3. Путевые и вычислительные процедуры (расписание) [3].

В составе ДВОР реализована прямоугольная укладка кадра в решетку вычислительных узлов и построение расписания путевых и вычислительных процедур.

Определим граф вычислений программы следующим образом. В качестве вершин графа вычислений программы будем рассматривать операции программы. Операции будем рассматривать арифметические, булевы, а также операции считывания из памяти и записи в память. Дуга соединяет пару операций, если вторая операция использует результат вычисления первой операции.

Ясно, что операции считывания из памяти данного, обозначенного безындексной переменной, соответствует источник на графе (т.е. такая вершина, в которую не входит ни одна дуга). А операции записи в память соответствует сток (т.е. такая вершина, из которой не выходит ни одна дуга).

Таким образом, граф вычислений содержит информацию о порядке выполнения арифметических операций, а также операций обращения к памяти.

Этот граф позволяет проследить порядок использования данных при вычислении выражений и применяется для формирования задания конвейеру, например, на суперкомпьютерах со структурно-процедурной организацией вычислений [5].

Рабочие определения

Ярусное представление графа - разбиение вершин графа на непересекающиеся подмножества (ярусы) Vi (i=1,…,L), при котором концевые вершины каждой дуги лежат в разных ярусах. Будем считать, что ярусы пронумерованы в «топологическом» порядке, т.е. для каждой дуги номер яруса начальной вершины меньше, чем номер яруса конечной вершины.

Ширина ярусного представления - максимальная мощность яруса.

Высота ярусного представления - количество ярусов.

Моноширинное ярусное представление - ярусное представление, в котором мощности всех ярусов одинаковы.

Короткая дуга - дуга, вершины которой лежат в соседних ярусах. Дуги, не удовлетворяющие такому условию, будем называть *длинными*.

Основной алгоритм прямоугольной укладки кадра

Описан основной алгоритм, который в процессе работы вызывает вспомогательные алгоритмы:

1. Для заданного графа кадра получить моноширинное ярусное представление (ЯП) без длинных дуг.
2. Пусть ЯП имеет ширину nGraphWidth и высоту nGraphHight.
   Если размеры ЯП превышают размеры решетки, то сообщить о невозможности уложить граф в решетку и прекратить работу.
3. Выделить в решетке прямоугольный фрагмент размера nGraphWidth × nGraphHight. Построить укладку ЯП в данный фрагмент с помощью алгоритма ПрямоугольнаяУкладкаЯП (будет рассмотрен ниже) [3].

Алгоритм построения и укладки моноширинного ярусного представления без длинных дуг

На входе: граф G(V,E).

На выходе: моноширинное ярусное представление (ЯП), не содержащее длинных дуг. Полученное ярусное представление соответствует графу G', такому, что исходный граф G является подграфом G'. Добавленные вершины помечаются знаками двух видов: «Т» (транзитная вершина, которая не выполняет операцию, а просто пересылает полученные данные дальше) и «П» (пустая вершина, ничего не выполняет, добавляется для удобства дальнейшей работы).

Алгоритм:

1. Построить ярусное представление графа. Например, с помощью итерационного процесса:
   1. Найти все источники(стоки).
   2. Поместить их в новый ярус.
   3. Удалить все источники(стоки) из графа.
   4. Если граф не пуст, то перейти к п.1
2. Каждую длинную дугу представить в виде цепи из новых вершин, имеющих пометку «Т». Допустим, дуга ведёт из яруса с номером i к ярусу с номером j, $j>i+1$. Преобразуем эту дугу в путь длины $(j–i)$, добавив $(j–i–1)$ новую вершину. Все вершины пути размещаются в ярусах последовательно. В результате одна длинная дуга преобразуется в несколько коротких.
3. Вычислить ширину полученного ярусного представления. Дополнить каждый ярус до данной величины (т.е. чтобы мощность всех ярусов была одинакова), добавив необходимое количество новых вершин с пометками «П».
4. Каждый ярус ЯП отображается в соответствующую колонку решетки, т.е. вершины i-го яруса отображаются в узлы i-й колонки вычислительных узлов. Вершины одного яруса размещаются в узлах колонки таким образом, чтобы минимизировать максимальное время пересылки данных. Требуемые пересылки определяются индексами узлов, в которых размещены начальная и конечная вершина каждой дуги (при этом начальная вершина размещается в узле предыдущей колонки, а конечная — в узле текущей колонки). Например, рассмотрим дугу, ведущую из вершины u яруса с номером i в вершину v яруса с номером $(i+1)$. И допустим, что отображение вершин в узлы задаётся функцией f, то есть вершина u размещается в узле с номером $f(u)$ колонки i, а вершина v — в узле с номером $f(v)$ колонки $i+1$. В этом случае время пересылки данных для дуги (u,v) характеризуется величиной $|f(u)–f(v)|$.
   Качество укладки характеризуется максимальным временем пересылки по всем дугам, то есть величиной $\max\{|f(u)–f(v)|: (u,v)\in E\}$ («стоимость» размещения). Для минимизации этой величины можно использовать поиск локального оптимума на основе следующих принципов:
   - последовательно размещать ярусы;
   - для каждого яруса находить такое размещение вершин этого яруса в узлах соответствующей колонки, при котором минимизируется $\max|f(u)–f(v)|$ по всем дугам, ведущим с предыдущего яруса в текущий(при этом считать, что размещение вершин предыдущих ярусов фиксировано).

В этом случае при размещении каждого яруса придется решать вариант классической NP-трудной задачи [3].

Оптимизация прямоугольной укладки

Совмещение вершин

Если две смежные вершины графа вычислений соответствуют операциям, которые могут быть выполнены одновременно, то такие две вершины следует «склеить» в одну. При совмещении вершин все дуги, которые входили в старые вершины (исходили из старых вершин), будут входить в новую (исходить из новой). Две операции можно выполнять одновременно в том случае, если они «разнотипные». Типы операций:

1. Преобразование данных (арифметическая операция).

2. Чтение данных (вершина, помеченная именем переменной, которая в рассматриваемом фрагменте программы только читается) или запись данных (вершина, соответствующая вхождению-генератору).
3. Транзит, т.е. передача значения.
4. Пустая вершина (добавленная для удобства работы) [3].

Подтягивание вершин

Преобразование заключается в укорачивании длинных дуг.

Пусть ширина построенного ярусного представления равна nWidth. Тогда следует перебрать все вершины графа, и для каждой вершины *v*, из которой выходят только длинные дуги, выполнить операцию «подтягивание вершины»:

1. Пусть вершина *v* находится на ярусе с номером *i* и из неё исходят только длинные дуги. Если мощность (*i*+1)-го яруса меньше, чем nWidth, то переместить вершину *v* в ярус (*i*+1).
2. Если все дуги, исходящие из *v*, всё ещё являются длинными, то продолжить (вернуться к п.1). Иначе операция завершена.

В результате выполнения этой операции некоторые короткие дуги могут стать длинными. Поэтому «подтягивание вершин» следует выполнять до тех пор, пока есть хотя бы одна вершина, для которой эта операция может быть выполнена [3].

Совмещение ярусов

В том случае, если мощность яруса с номером *i* меньше максимальной(nWidth), возможно совмещение этого яруса с соседним — (*i*–1)-м или (*i*+1)-м. То есть все вершины яруса (или часть вершин, в зависимости от количества «свободных мест» в соседнем ярусе) перемещаются в соседний ярус. В результате полученное ЯП может перестать быть «ярусным» в смысле строгого определения — если появятся дуги, соединяющие вершины из одного и того же яруса. Но полученное «квазиярусное» представление всё равно можно будет уложить в решётку рассмотренным ранее алгоритмом [3].

Заключение

В статье обсуждается проблема разбиения программы на кадры и отображение кадров на архитектуру суперкомпьютеров. Данная проблема является актуальной в настоящее время, так как ее решение позволит эффективно отображать программы на архитектуру компьютера и выполнять в них параллельные вычисления. В составе Диалогового высокоуровнего оптимизирующего распараллеливателя реализован модуль разбиения программы на кадры.

Также приведена схема алгоритма прямоугольной укладки кадра в решетку вычислительных узлов и способы ее оптимизации. Алгоритм и оптимизирующие его преобразования программно реализованы в составе Диалогового высокоуровнего оптимизирующего распараллеливателя.

ЛИТЕРАТУРА:

1. Корнеев В.В. Проблемы программирования суперкомпьютеров на базе многоядерных мультитредовых кристаллов. Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийской суперкомпьютерной конференции (21-26 сентября 2009 г., г.Новороссийск). – М.: Изд-во МГУ, 2009. – 524 с.
2. Левин И.И. Структурно-процедурная организация параллельных вычислений // Материалы Четвертой Международной научной молодежной школы "Высокопроизводительные вычислительные системы". - Таганрог: Изд-во ТТИ ЮФУ, 2007. - С. 49-68.
3. Адигеев М.Г. Укладка графа вычислений в двумерную решетку. http://ops.rsu.ru/download/works/Calc2mesh.pdf Дата обращения 13 июля 2012 г.
4. Корнеев В.В. Архитектура вычислительных систем с программируемой структурой. Новосибирск: Наука, 1985.
5. Математические методы распараллеливания рекуррентных циклов для суперкомпьютеров с параллельной памятью. – Ростов-/Д:Изд-во Рост. Ун-та, 2004. – 192 с.