Тема 1: Распараллеливание выражений на примере арифметических. Основные характеристики сложности и параллельности

Что подлежит распараллеливанию?

1.Задача (декомпозиция на подзадачи меньшей размерности)
2.Метод решения
3.Алгоритм
4.Программа
5.Циклы
6.Выражения

Вопросы распараллеливания арифметико-логических выражений и рекурсивных схем, а также распараллеливания циклов - основы построения распараллеливающих трансляторов.
На сегодняшний день эти проблемы наиболее изучены в параллельном программировании

Вычисление выражения, с точки зрения структуры – ациклический процесс (дерево), который может быть реализован ациклической схемой программы.

Основная цель распараллеливания АВ – разработать дерево вычислений минимальной высоты (т.е. выражение должно вычисляться за минимально возможное время или число шагов алгоритма при условии реализации на каждом ярусе ***максимального*** числа независимых операций). Распараллеливание АВ осуществляется в рамках модели **MIMD** с неограниченным параллелизмом.

***Характеристики сложности вычисления АВ*:**

- ***время вычисления АВ* t –** число ярусов (высота) дерева вычислений;
- ***число операций* w** – общее число операций при вычислении АВ;
- ***операции, лежащие на одном ярусе*** – ширина яруса, определяет требуемое число процессоров;
- требуемое ***число вычислителей* p *(или процессоров)*** – максимальная ширина яруса дерева вычислений.

Задача распараллеливания выражений – построить алгоритм, который по каждому выражению **E** дает эквивалентное ему $\mathbf{E_p}$ с минимальной высотой дерева вычислений.

✓ Два выражения E и E_p называются ***эквивалентными***, если выражение E преобразуется в E_p и, наоборот, путем использования

конечного числа раз законов ассоциативности, коммутативности и дистрибутивности.

Характеристики параллельности:

- $S_p(n) = T_1(n) / T_p(n)$ - ***ускорение*** параллельного алгоритма,
- $Q_p(n) = S_p(n) / p$ - ***эффективность*** параллельного алгоритма,

где n – *число параметров задачи (для АВ – число различных переменных, входящих в АВ),*

$T_p(n)$ - *время выполнения параллельного алгоритма на вычислительной системе с числом процессоров p>1,*

$T_1(n)$ - *время работы "наилучшего" последовательного алгоритма.*

<u>Чем ближе значение</u> $S_p(n) \to p$, **а** $Q_p(n) \to 1$, ***<u>тем "лучше" построенный параллельный алгоритм.</u>***

Пример 1:

Рассмотрим выражение E = (x + (a * ((b / c) * d))) - (y - z) и представим процесс его вычисления в виде дерева (см. Рис.1.).

Дерево вычислений одного из возможных эквивалентных для E выражений E_p= ((a * b) / (c / d)) – ((y – z) – x)) имеет вид (см. **Рис.2.**):

Рис.1.

Рис.2.

<u>Характеристики сложности</u> <u>(рис.1, рис.2):</u>

для выражения E:

t = 5,
p = 2,
w = 6,

для выражения E_p:

t_p = 3,
p = 3,
w_p = 6.

<u>Характеристики параллельности</u> <u>(рис.1, рис.2):</u>

для выражения E:

$T_1(n) = n - 1 = 6$,

для выражения E_p:

$T_1(n) = n - 1 = 6$,

$T_p(n) = t = 5$, $T_p(n) = t_p = 3$

$S_p(n) = 1.2$, $S_p(n) = 2$,

$Q_p(n) = 0.6$ $Q_p(n) = 0.66$.

Таким образом, выражение E_p имеет лучшие характеристики параллельности, чем выражение E.

Лемма Брента: Если при неограниченном числе процессоров время выполнения параллельной схемы вычисления выражения, состоящего из w операций, может быть выполнено за время ***t***, то при наличии ограниченного числа ***p'*** процессоров время выполнения составит не более ***t'=t+(w-t)/p'***.

Рекурсивные схемы

1. Рекурсия

Важнейшим классом арифметических выражений являются рекурсивные выражения. Рекурсия встречается в матричных операциях, является основой многих методов для вычисления значений функции и используется в ряде методов численного интегрирования.

При рекурсивных операциях значение каждого очередного члена последовательности или элемента структуры зависит от значений одного или нескольких предшествующих членов.

В последовательных компьютерах рекурсия реализуется в виде циклических процессов. Рекурсия встречается не только при числовой обработке, но и при обработке сигналов, операциями над символами и т.п. Рекурсивный характер обработки накладывает ограничения на возможности распараллеливания, которое может быть достигнуто лишь реорганизацией последовательности обработки.

2. Распараллеливание рекурсивных схем

1. Остановимся на простейшем примере линейной рекурсии первого порядка для вычисления суммы $S = \sum_{i=1}^{n} x_i$ всех элементов вектора $X = \{x_i\}$, где i =1, 2, …,n . В общем случае, в качестве операции может выступать любая ассоциативная операции. К числу таких операций относятся сложение, умножение, поиск максимума и минимума среди вещественных чисел, операции сложения и умножения матриц.

Для нахождения этой суммы можно воспользоваться следующим рекурсивным соотношением:

$S_i = S_{i-1} + x_i$, i =1, 2,…,n.

Прямая реализация этого выражения выполняется последовательностью машинных команд.

Эта последовательная реализация процесса вычисления иллюстрируется информационным графом на рис. 3, и поскольку каждый из операндов используется однократно, такой граф представляет собой бинарное дерево. Если выполнение каждой примитивной операции занимает время t, то суммарные затраты времени (при высоте графа $q=n-1$) реализации этого графа составят

$$T_1(n) = t * (n - 1) = t * q.$$

Очевидно, что при наличии нескольких вычислителей алгоритм будет неэффективным, и задача состоит в том, чтобы минимизировать высоту дерева. Преобразование выполняется в соответствии с ***законами коммутативности, ассоциативности и дистрибутивности***, в результате чего может быть получен трансформированный граф, показанный на рис. 4.

Высота такого дерева $q' = [\log_2 n]$, а длительность реализации составит $T_p = t * [\log_2 n]$ в предположении, что операции суммирования реализуются за то же время t и затраты времени на обмен между отдельными вычислителями эквивалентны затратам времени на обмен между ячейкой памяти и вычислителем в последовательной структуре. Такой метод нахождения суммы элементов вектора получил название ***метода рекурсивного сдваивания*** или ***метода нахождения параллельных каскадных сумм***.

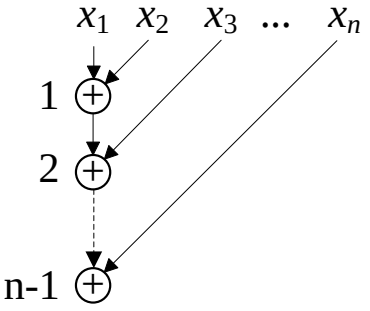

Рис. 3. Прямое суммирование

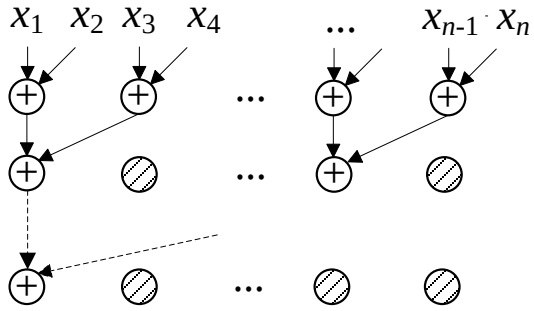

Рис. 4. Попарное суммирование

Максимальное ускорение $\xi_p = T_1 / T_p$ можно оценить, как:

$$O(n/\log_2 n).$$

Оно достигается при числе вычислителей не менее **n/2**, причем полностью все вычислители загружены только *на первом шаге вычислений*, на последнем же шаге используется лишь один вычислитель.

Не используемые на каждом шаге вычислители можно отразить на графе в виде «пустых» операторов (заштрихованных кругов на рис. 4). Тогда эффективность построенной схемы алгоритма можно оценить, как отношение числа действительных к общему числу операторов (действительных и пустых). Очевидно, что такое определение понятия эффективности сродни понятию коэффициента полезного действия. Для рассматриваемого алгоритма оно составляет **0.5** и не зависит от **n**.

Более подробный анализ рассмотренных выражений см. слайд: Метод сдваивания.ppsx

Рассмотренный принцип рекурсивного сдваивания справедлив для произвольной ассоциативной бинарной операции на произвольном множестве.

2. Более общим случаем линейной рекурсии является вычисление значения полинома по схеме Горнера:

$$S = (\ldots(d_0 a_1 + d_1)\cdot a_2 + d_2)\cdot a_3 + d_3)\ldots)\cdot a_n + d_n).$$

Отличительной особенностью схемы является чередование аддитивных (сложение, вычитание) и мультипликативных (умножение, деление, обратное деление) операций. Подобные выражения называют альтернированными.

Рекурсивный алгоритм для реализации этой схемы имеет вид:

$$S_i = S_{i-1}\cdot a_i + d_i, \quad i = 1,2,\ldots,n \text{ и } S_0 = d_0.$$

Последовательная программа для нахождения S по этой схеме подобна приведенной выше для нахождения суммы, лишь команде сложения должна предшествовать команда умножения. Схема Горнера и рекурсивный алгоритм ее реализации весьма удобны для последовательных вычислений, однако их непосредственная реализация в параллельных вычислениях не приводит к ускорению, так как для выполнения очередной операции необходим результат, полученный на предыдущем шаге.

Известно, что альтернированные выражения всегда могут быть представлены путем разложения в виде функции нескольких арифметических выражений, например, в виде:

$$S = A + B,$$

где каждое арифметическое выражение A и B содержит не более n арифметических операций, что доказывает возможность построения более быстрого параллельного алгоритма.

Поскольку основным ограничением для параллельной реализации рассматриваемой рекурсии является зависимость текущей операции от результата предыдущей, преобразуем схему вычислений так, чтобы вычисляемое значение S_i на текущем шаге было связано с результатом, полученным не на предыдущем шаге, а на один шаг раньше, т.е. с S_{i-2}:

$$S_i = S_{i-1}\cdot a_i + d_i,$$
$$S_{i-1} = S_{i-2}\cdot a_{i-1} + d_{i-1}.$$

После подстановки второго выражения в первое получим:

$$S_i = (S_{i-2}\cdot a_{i-1} + d_{i-1})\cdot a_i + d_i = S_{i-2}\cdot a_{i-1}\cdot a_i + d_{i-1}\cdot a_i + d_i.$$

Обозначив в этом выражении через

$a_i^{(1)} = a_i \cdot a_{i-1}$ и

$d_i^{(1)} = d_{i-1} \cdot a_i + d_i$,

где верхний индекс означает номер последовательно применяемого описанного преобразования, получим:

$$S_i = S_{i-2} \cdot a_i^{(1)} + d_i^{(1)}.$$

Последнее выражение также является рекурсией, и к нему вновь можно применить это преобразование:

$$S_{i-2} = S_{i-4} \cdot a_{i-2}^{(1)} + d_{i-2}^{(1)},$$

$$S_i = (S_{i-4} \cdot a_{i-2}^{(1)} + d_{i-2}^{(1)}) \cdot a_i^{(1)} + d_i^{(1)} = S_{i-4} \cdot a_i^{(2)} + d_i^{(2)},$$

где $a_i^{(2)} = a_i^{(1)} \cdot a_{i-2}^{(1)}$ и $d_i^{(2)} = d_{i-2}^{(1)} \cdot a_i^{(1)} + d_i^{(1)}$.

Последовательное применение указанного преобразования носит название циклической редукции, на каждом шаге которой получаем:

$$a_i^{(k)} = a_i^{(k-1)} \cdot a_{i-2^{k-1}}^{(k-1)} \text{ и}$$

$$d_i^{(k)} = d_{i-2^{k-1}}^{(k-1)} \cdot a_i^{(k-1)} + d_i^{(k-1)},$$

где k – номер шага редукции.

В результате применения $\log_2 n$ шагов получим:

$S = A + B$, где

$$A = a_n a_{n-1} \ldots a_1 d_0,$$

$$B = a_n a_{n-1} \ldots a_2 d_1 + a_n a_{n-1} \ldots a_3 d_2 + \ldots + d_n.$$

Таким образом, нахождение значения полинома сводится к нахождению каскадной суммы произведений.

Параллельный алгоритм для вычисления коэффициентов $a_i^{(k)}$ аналогичен рассмотренному выше для нахождения сумм элементов вектора (операция сложения заменяется на операцию умножения).

Рассмотрим пример вычисления схемы Горнера для n=4.

$$S = (((d_0 a_1 + d_1) \cdot a_2 + d_2) \cdot a_3 + d_3) \cdot a_4 + d_4,\ S_0 = d_0.$$

Используя формулы для циклической редукции, с учетом введенных выше обозначений, имеем:

$$S_4 = S_2 \cdot a_4^{(1)} + d_4^{(1)} = S_2 \cdot a_3 \cdot a_4 + d_3 \cdot a_4 + d_4,$$

$$S_2 = S_0 \cdot a_2^{(1)} + d_2^{(1)} = d_0 \cdot a_1 \cdot a_2 + d_1 \cdot a_2 + d_2.$$

Для данного примера дерево параллельного вычисления может быть представлено в следующем виде (см. рис. 5):

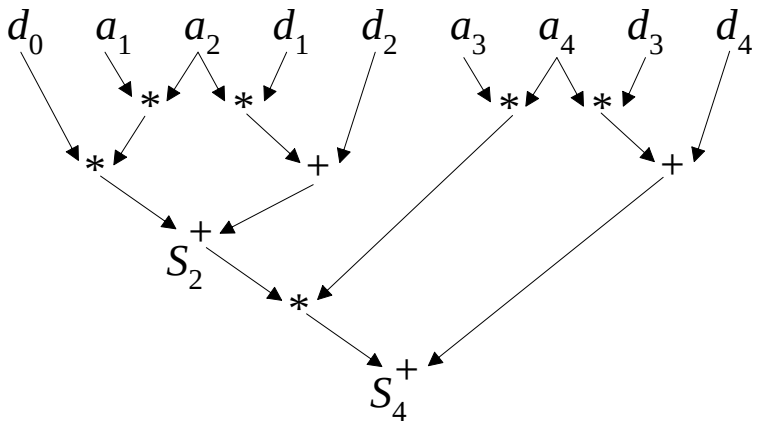

Рис.5. Дерево параллельного вычисления схемы Горнера для S_4

Все арифметические операции сгруппированы по уровням; однотипные операции умножения или сложения, находящиеся в отношении безразличия, могут выполняться параллельно.

Характеристики сложности и параллельности для рассмотренного **примера**:

$w_p=10$
$p=4$
$T_1(L)=8$
$T_p(L)=5$
$S_p(L)=8/5=1,6$
$Q_p(L)=1,6/4=0,4$