Параллельное программирование

Анализ эффективности параллельных вычислений

11.05.2016 (продолжение лекции)

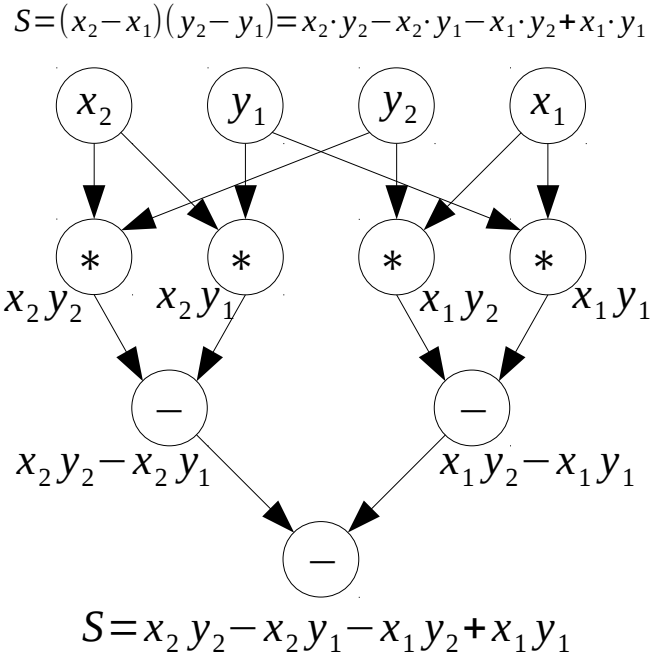
Анализ эффективности параллельных вычислений

Цель анализа эффективности параллелизма – получить оценку ускорения процесса вычислений. Для этого предварительно создаётся модель вычислений, обычно в виде графа "операции-операнды", который описывает информационные зависимости в алгоритме задачи.

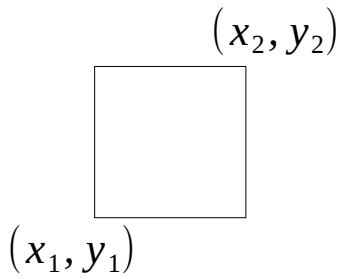
Будем предполагать, что время выполнения всех вычислительных операций является одинаковым и равно некоторой условной единице и что передача данных между вычислительными устройствами осуществляется мгновенно, то есть без каких-либо затрат времени.

Множество всех операций, выполняемых в алгоритме решения задачи, и существующие между операциями информационные зависимости представим в виде ациклического ориентированного графа, который обозначим как $G(V,R)$.

Пример



S равно площади прямоугольника, когда имеются диагональные координаты



Могут быть и другие схемы вычислений. Разные схемы вычислений обладают разными возможностями для распараллеливания. Таким образом, можно ставить задачу выбора наиболее подходящей для распараллеливания схемы.

Вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг – для операций вывода.

Обозначим:

- $\overline{V}$ – множество вершин графа без вершин ввода
- $d(G)$ – диаметр графа

Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены параллельно.

Для описания параллельного выполнения алгоритма введём обозначения:

Пусть p – количество процессоров, используемых для выполнения алгоритма.

Для параллельного выполнения вычислений зададим множество $H_p=\{(i,P_i,t_i) \mid i\in V\}$, которое будем называть расписанием. Тройка (i,P_i,t_i) для каждой операции $i\in V$ указывает номер используемого для операции процессора P_i и время начала операции t_i.

Для того, чтобы расписание было реализуемым, необходимо при задании множества H_p потребовать выполнение следующих условий:

1. $\forall i,j\in V:\ t_i=t_j \Rightarrow P_i\neq P_j$ (если операции выполняются в один момент времени, то поручаются разным процессорам)
2. $\forall (i,j)\in R \Rightarrow t_j\geq t_i+1$ (к назначенному моменту выполнения операции все необходимые данные уже должны быть вычислены)

Вычислительная схема алгоритма, представленная в виде графа G, совместно с расписанием H_p может рассматриваться как модель параллельного алгоритма $A_p(G, H_p)$, исполняемого с использованием P процессоров. Время выполнения параллельного алгоритма определяется максимальным значением времени, используемым в расписании.

$$T_p(G, H_p) = \max_{i \in V} (t_i + 1)$$

Для выбранной схемы вычислений желательно использовать расписание, обеспечивающее минимальное время исполнения алгоритма.

$$T_p(G) = \min_{H_p} T_p(G, H_p)$$

Уменьшение времени выполнения может быть обеспечено и путём подбора наилучшей вычислительной схемы.

$$T_p = \min_{G} T_p(G)$$

Оценки $T_p(G, H_p)$, $T_p(G)$ и T_p могут быть использованы в качестве показателя времени исполнения параллельного алгоритма. Кроме того, для анализа максимально возможного параллелизма можно определить оценку наиболее быстрого исполнения алгоритма $T_\infty = \min_{P \geq 1} T_p$.

Увеличение числа процессоров не всегда ведёт к уменьшению времени исполнения алгоритма.

T_∞ можно рассматривать как минимально возможное время выполнения алгоритма при использовании неограниченного количества процессоров. Концепция вычислительной системы с бесконечным количеством процессоров называется паракомпьютером и широко используется при теоретическом анализе параллельных вычислений.

Ясно, что T_1 определяет время выполнения алгоритма при использовании одного процессора и соответствует времени выполнения последовательного варианта алгоритма решения задачи.

Оценка T_1 важна при определении эффекта от распараллеливания.

Ясно, что $T_1(G) = |\overline{V}|$ – количество вершин в вычислительной схеме G без вершин ввода.

Вообще говоря, если существует несколько различных записей последовательного алгоритма (несколько схем G), то в качестве G принимают наименьшее время по всем таким схемам:

$T_1 = \min_G T_1(G)$.

Если существует несколько алгоритмов, то $T_1^* = min\, T_1$ (выбираем алгоритм с наименьшим временем).

Свойства оценок времени выполнения параллельного алгоритма

Приведём теоремы, характеризующие свойства оценок времени выполнения параллельного алгоритма (без доказательств).

Теорема 1

Минимально возможное время выполнения параллельного алгоритма определяется длиной максимального пути вычислительной схемы алгоритма.

$T_\infty(G) = d(G)$

Теорема 2

Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из любой вершины ввода, и пусть входная степень вершины схемы не превышает двух. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу значением $T_\infty(G) \geq \log_2 n$, где n – количество вершин ввода в схеме алгоритма.

Теорема 3

При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров.

$\forall g = c\,p, (0 < c < 1) \Rightarrow T_p \leq c\,T_g$

Теорема 4

Для любого количества используемых процессоров справедлива следующая верхняя оценка времени выполнения параллельного алгоритма:

$\forall p: \; T_p \leq T_\infty + \frac{T_1}{p}$

Теорема 5

Времени выполнения алгоритма, которое сопоставимо с минимально возможным временем T_∞, можно достичь при количестве процессоров $p \sim \frac{T_1}{T_\infty}$.

А именно, $p \geq \frac{T_1}{T_\infty} \Rightarrow T_p \leq 2T_\infty$.

При меньшем количестве процессоров время выполнения алгоритма не может превышать более чем в два раза наилучшее время вычислений при имеющемся числе процессоров.

То есть: $p < \frac{T_1}{T_\infty} \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}$

25.05.2016

Приведённые утверждения позволяют дать следующие рекомендации по правилам формирования параллельных алгоритмов:

1. При выборе вычислительной схемы алгоритма должен использоваться граф с минимально возможным диаметром (теорема 1).
2. Для параллельного выполнения целесообразное количество процессоров определяется величиной $p \sim \frac{T_1}{T_\infty}$.
3. Время выполнения параллельного алгоритма ограничивается сверху величинами, приведёнными в теоремах 4 и 5.

Для выбора рекомендаций по формированию расписания параллельного выполнения алгоритма приведём доказательство теоремы 4:

Обозначим через H_∞ расписание для достижения минимально возможного времени выполнения T_∞. Для каждой итерации τ ($0 \leq \tau \leq T_\infty$) выполнения расписания H_∞ обозначим через n_τ количество операций, выполняемых в ходе итерации τ.

Расписание выполнения алгоритма с использованием p процессоров может быть построено следующим образом: разделим выполнение алгоритма на T_∞ шагов. На каждом шаге τ следует выполнять все n_τ операций, которые выполнялись на итерации τ в расписании H_∞. Выполнение этих операций может быть выполнено не более чем за $\left\lceil \frac{n_\tau}{p} \right\rceil$ итераций при использовании p процессоров. В результате время выполнения T_p может быть оценено следующим образом: $T_p = \sum_{\tau=1}^{T_\infty} \left\lceil \frac{n_\tau}{p} \right\rceil < \sum_{\tau=1}^{T_\infty} \left(\frac{n_\tau}{p} + 1 \right) = \frac{T_1}{p} + T_\infty.$

ЧТД

Это доказательство даёт практический способ построения расписания параллельного алгоритма. Первоначально расписание может быть построено для паракомпьютера (компьютер с неограниченном числом процессоров), а затем по схеме вывода теоремы может быть построено расписание для конкретного числа процессоров.

Показатели эффективности параллельного алгоритма

1. Ускорение $S_p(n)=\frac{T_1(n)}{T_p(n)}$, n – некоторый показатель, отражающий вычислительную сложность решаемой задачи. В качестве n можно, например, задать количество входных данных для рассматриваемой задачи.
2. Эффективность $E_p(n)=\frac{T_1(n)}{T_p(n)\cdot p}=\frac{S_p(n)}{p}$.

   Величина эффективности определяет среднюю долю времени выполнения алгоритма, в течение которой процессоры реально используются для решения задачи.

Ясно, что в наилучшем случае $S_p(n)=p$, $E_p(n)=1$. При практическом применении этих показателей для оценки эффективности параллельных вычислений следует учитывать два важных момента:

1. При определённых обстоятельствах ускорение может оказаться больше числа используемых процессоров (то есть $S_p(n)>p$). В этом случае говорят о существовании сверхлинейного ускорения.

Одной из причин такого ускорения может быть неравноправность выполнения последовательной и параллельной программы.

Например, при решении задач на одном процессоре оказывается недостаточно оперативной памяти для хранения всех обрабатываемых данных, и в результате становится необходимо использование более медленной внешней памяти. А в случае нескольких процессоров оперативной памяти может оказаться достаточно за счёт разделения данных между процессорами.

Другая причина сверхлинейного ускорения – нелинейный характер зависимости сложности решения задачи от объёма обрабатываемых данных.

Пример: алгоритм пузырьковой сортировки характеризуется квадратичной зависимостью количества необходимых операций числа упорядочиваемых данных. При распределении сортируемого массива между несколькими процессорами может быть получено ускорение, превышающее число процессоров.

Ещё одна причина сверхлинейного ускорения – различие вычислительных схем

последовательного и параллельного метода.

2. Попытки повышения качества параллельных вычислений по одному из показателей (ускорение либо эффективность) могут привести к ухудшению ситуации по другому показателю, так как эти показатели противоречивы. Поэтому необходимо выбирать некоторый компромиссный вариант.

При выборе параллельного способа решения задачи может оказаться полезной оценка стоимости вычислений, которая определяется как $C_p = p \cdot T_p$. В связи с этим можно определить понятие стоимостно оптимального параллельного алгоритма как метода, стоимость которого является пропорциональной времени выполнения наилучшего последовательного алгоритма.

Пример.

Вычисление частных сумм последовательности числовых значений.

$$S_k = \sum_{i=1}^{k} x_i, \quad 1 \le k \le n$$

n – количество элементов последовательности.

Рассмотрим сначала вычисление общей суммы $S = \sum_{i=1}^{n} x_i$. Как известно, алгоритм последовательного суммирования представляет собой следующее:

```
S := 0
S := S + x1
S := S + x2
...
```

Вычислительная схема последовательного алгоритма:

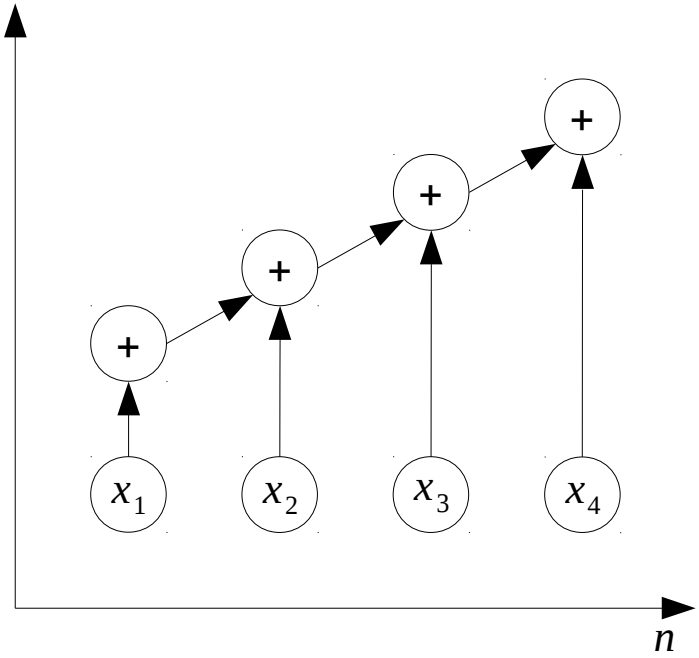


Рис. 2 (начинаем нумерацию рисунков со второго)

Каскадная схема суммирования

Существует *каскадная схема суммирования*, которая состоит в том, что на первой итерации все исходные данные разбиваются на пары, для каждой пары вычисляется сумма и далее все полученные суммы также разбиваются на пары и происходит суммирование пар. И так далее.

Схема каскадного суммирования:

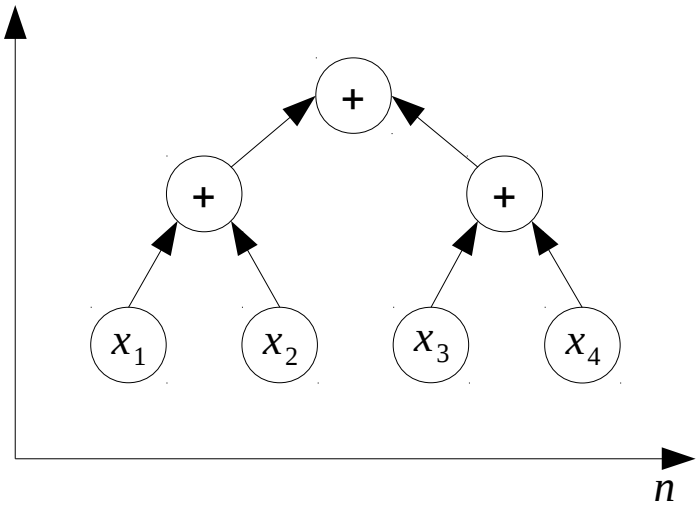


Рис. 3

Количество итераций каскадной схемы: $k = \log_2 n$.

Общее число операций суммирования для последовательного алгоритма:

$R_{посл}=\frac{n}{2}+\frac{n}{4}+...+1=n-1$.

Число операций каскадной схемы и число операций последовательного алгоритма суммирования совпадают.

Параллельное исполнение отдельных итераций каскадной схемы

Общее количество выполнения последовательных операций совпадает с последовательным: $k_{пар.}=\log_2 n$.

Если обозначим $T_1=k_{посл.}$, $T_p=k_{пар.}$, то получим:

Ускорение: $S_p=\frac{T_1}{T_p}=\frac{n-1}{\log_2 n}$

Эффективность: $E_p=\frac{n-1}{p\cdot T_p}=\frac{n-1}{p\log_2 n}=\frac{n-1}{\frac{n}{2}\log_2 n}$, где $\frac{n}{2}$ – количество процессоров, необходимых для каскадной схемы.

Время параллельного выполнения каскадной схемы совпадает с оценкой для паракомьютера в теореме 2, но эффективность использования процессоров уменьшается при увеличении числа процессоров. $\lim\limits_{n\to\infty} E_p\to 0$.

Модифицированная каскадная схема

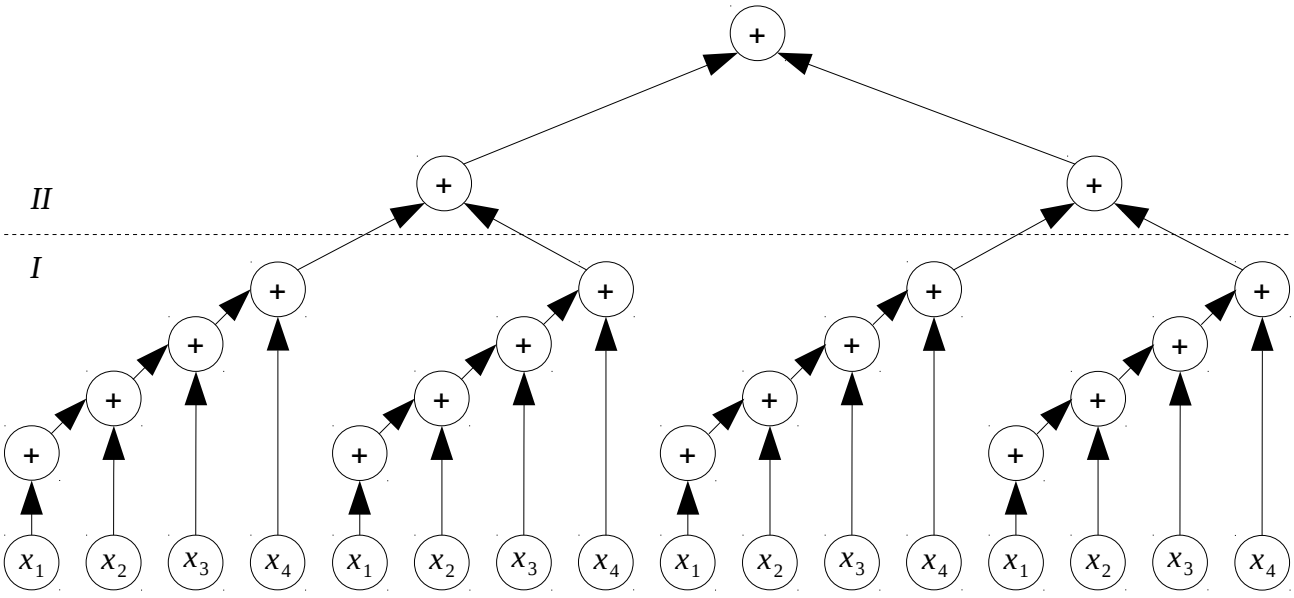


Рис. 4

Для упрощения получения оценок предположим, что $n=2^k$, $k=2^s$. Тогда все проводимые вычисления подразделяются на два последовательно выполняемых этапа суммирования (I, II на рисунке).

На первом этапе все суммируемые значения подразделяются на $\frac{n}{\log_2 n}$ групп, в каждой из которых содержатся $\log_2 n$ элементов. Для каждой группы вычисляется сумма значений по схеме последовательного алгоритма. Вычисления в каждой группе могут выполняться независимо, то есть параллельно. Для этого понадобится $\frac{n}{\log_2 n}$ процессоров.

На втором этапе для полученных сумм отдельных групп применяется обычная каскадная схема.

Оценки:

I. Для выполнения первого этапа количество выполняемых итераций составляет $\| \log_2 n$, количество процессоров $p_1=\frac{n}{\log_2 n}$.

II. Для второго этапа количество параллельных ... $\| \log_2\left(\frac{n}{\log_2 n}\right)\le\log_2 n$, количество процессоров $p_2=\frac{n}{2\log_2 n}$.

Таким образом для данной схемы получаются следующие оценки:

$T_p=2\log_2 n$; $p=\frac{n}{\log_2 n}$

Ускорение $S_p=\frac{T_1}{T_n}=\frac{n-1}{2\log_2 n}$

Эффективность $E_p=\frac{T_1}{pT_p}=\frac{n-1}{2\frac{n}{\log_2 n}\cdot\log_2 n}=\frac{n-1}{2n}$

Если сравнить полученные оценки с показателями обычной каскадной схемы, то можно увидеть, что ускорение уменьшилось в два раза, а для эффективности можно получить асимптотически ненулевую оценку снизу: $E_p\ge 0{,}25$. Если же рассмотреть предел: $\lim\limits_{n\to\infty} E_p=0{,}5$.

Замечание:

Данные значения показателей достигаются при количестве процессоров аналогично теореме

5.

Замечание 2:

Модифицированный каскадный алгоритм является стоимостно оптимальным, так как стоимость вычислений $C_p = p \cdot T_p = \frac{n}{\log_2 n} \cdot 2\log_2 n$ является пропорциональной времени выполнения последовательного алгоритма.

Вычисление частных сумм

Перейдём к рассмотрению задачи на вычисление частных сумм (то есть i-ый элемент частной суммы является суммой первых i исходных элементов).

Последовательный алгоритм суммирования содержит n операций, параллельное выполнение каскадной схемы в явном виде не приводит к желаемым результатам.

Схема параллельного вычисления всех частных сумм:

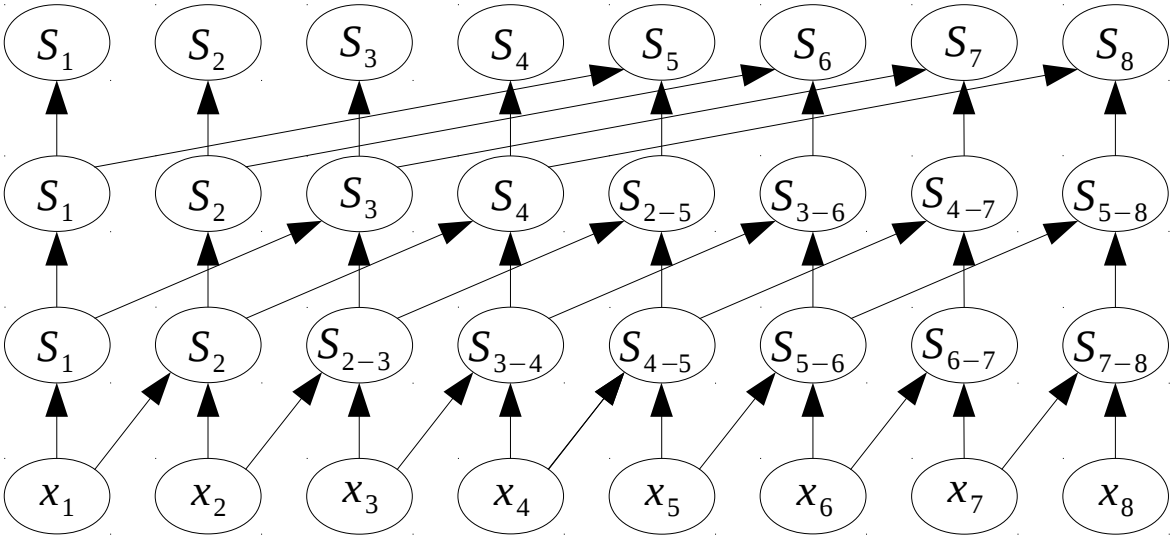


Рис. 5

S_{i-j} – сумма от i-го до j-го элемента последовательности.

Этот алгоритм обеспечивает получение результатов за $\log_2 n$ параллельных операций и состоит в следующем:

Перед началом суммирования создаётся копия вектора суммированных значений в переменной S (то есть $S = X$). Далее на каждой итерации суммирования i, где $1 \le i \le \log_n 2$, формируется вспомогательный вектор Q путём сдвига вправо вектора S на 2^{i-1} позиций (освобождающиеся при сдвиге позиции слева заполняются нулями). Итерация алгоритма завершается параллельной операцией суммирования векторов S и Q.

Всего параллельных операций сложения: $\log_2 n$, на каждой итерации алгоритма выполняется параллельно n скалярных операций сложения. Общее число скалярных операций $k_{пар.} = n \cdot \log_2 n$. Параллельный алгоритм содержит большее число операций, чем последовательный.

Необходимое количество процессоров $p = n$, то есть совпадает с числом суммируемых значений.

Ускорение: $S_p = \frac{T_1}{T_p} = \frac{n}{\log_2 n}$

Эффективность: $E_p = \frac{T_1}{p T_p} = \frac{n}{p \log_2 n} = \frac{n}{n \log_2 n} = \frac{1}{\log_2 n}$

Видно, что эффективность алгоритма уменьшается при увеличении числа суммируемых значений, поэтому алгоритм необходимо модифицировать примерно так, как он был модифицирован для обычной каскадной схемы.

Оценка максимально достижимого параллелизма

1. Закон Амдаля (Амдаль – Amdahl)

Комментарий:

Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчётов, которые не могут быть распараллелены.

Пусть f – доля последовательных вычислений в алгоритме, тогда ускорение при использовании p процессоров ограничивается величиной: $S_p \le \frac{1}{f + \frac{1-f}{p}} \le S^* = \frac{1}{f}$ – закон Амдаля.

Пример

При наличии всего 10% последовательных команд, выполняемых вычислением, эффект использования параллелизма не может превышать десятикратного ускорения обработки данных.

Если вернуться к примеру вычисления суммы значений, то для каскадной схемы доля последовательных расчётов составляет $f = \frac{\log_2 n}{n}$, следовательно $S^* = \frac{n}{\log_2 n}$.

(скопировано с документа ЕВ):

1. Закон Амдаля. Достижению максимального ускорения может препятствовать существование в выполняемых вычислениях последовательных расчетов, которые не могут быть распараллелены. Пусть f есть *доля последовательных вычислений* в применяемом алгоритме обработки данных, тогда в соответствии с *законом Амдаля* (*Amdahl*) ускорение процесса вычислений при использовании p процессоров ограничивается величиной

$$S_p \le \frac{1}{f + (1-f)/p} \le S^* = \frac{1}{f}.$$

Так, например, при наличии всего 10% последовательных команд в выполняемых вычислениях, эффект использования параллелизма не может превышать 10-кратного ускорения обработки данных. Для рассмотренного учебного примера вычисления суммы значений для каскадной схемы доля последовательных расчетов составляет $f = \log_2 n / n$ и, как результат, величина возможного ускорения ограничена оценкой $S^* = n / \log_2 n$.

Закон Амдаля характеризует одну из самых серьезных проблем в области параллельного программирования (алгоритмов без определенной доли последовательных команд практически не существует). Однако часто доля последовательных действий характеризует не возможность параллельного решения задач, а последовательные свойства применяемых алгоритмов. Как результат, доля последовательных вычислений может быть существенно снижена при выборе более подходящих для распараллеливания методов.

Следует отметить также, что рассмотрение закона Амдаля происходит при предположении, что доля последовательных расчетов f является постоянной величиной и не зависит от параметра n, определяющего вычислительную сложность решаемой задачи. Однако для большого ряда задач доля $f=f(n)$ является убывающей функцией от n, и в этом случае ускорение для фиксированного числа процессоров может быть увеличено за счет увеличения вычислительной сложности решаемой задачи. Данное замечание может быть сформулировано как утверждение, что ускорение $S_p = S_p(n)$ является возрастающей функцией от параметра n (данное утверждение часто именуется как *эффект Амдаля*). Так, например, для учебного примера вычисления суммы значений при использовании фиксированного числа процессоров p суммируемый набор данных может быть разделен на блоки размера n/p, для которых сначала параллельно могут быть вычислены частные суммы, а далее эти суммы можно сложить при помощи каскадной схемы. Длительность последовательной части выполняемых операций (минимально-возможное время параллельного исполнения) в этом случае составляет

$$T_p = (n/p) + \log_2 p,$$

что приводит к оценке доли последовательных расчетов как величины

$$f = (1/p) + \log_2 p / n.$$

Как следует из полученного выражения, доля последовательных расчетов f убывает с ростом n и в предельном случае мы получаемом идеальную оценку максимально возможного ускорения $S^*=p$.

2. Закон Густавсона - Барсиса. Оценим максимально достижимое ускорение исходя из имеющейся доли последовательных расчетов в выполняемых параллельных вычислениях:

$$g = \frac{\tau(n)}{\tau(n) + \pi(n)/p},$$

где $\tau(n)$ и $\pi(n)$ есть времена последовательной и параллельной частей выполняемых вычислений соответственно, т.е.

$$T_1 = \tau(n) + \pi(n),\ T_p = \tau(n) + \pi(n)/p.$$

С учетом введенной величины g можно получить

$$\tau(n) = g \cdot (\tau(n) + \pi(n)/p),\ \ \pi(n) = (1-g)p \cdot (\tau(n) + \pi(n)/p),$$

что позволяет построить оценку для ускорения

$$S_p = \frac{T_1}{T_p} = \frac{\tau(n) + \pi(n)}{\tau(n) + \pi(n)/p} = \frac{(\tau(n) + \pi(n)/p)(g + (1-g)p)}{\tau(n) + \pi(n)/p},$$

которая после упрощения приводится к виду *закона Густавсона-Барсиса* (*Gustafson-Barsis's law*) (см. Quinn (2004))

$$S_p = g + (1-g)p = p + (1-p)g.$$

Применительно к учебному примеру суммирования значений при использовании p процессоров время параллельного выполнения, как уже отмечалось выше, составляет

$$T_p = (n/p) + \log_2 p,$$

что соответствует последовательной доле

$$g = \frac{\log_2 p}{(n/p) + \log_2 p}.$$

За счет увеличения числа суммируемых значений величина g может быть пренебрежимо малой, обеспечивая получение идеального возможного ускорения $S_p = p$.

При рассмотрении закона Густавсона-Барсиса следует учитывать еще один важный момент. При увеличении числа используемых процессоров темп уменьшения времени параллельного решения задач может падать (после превышения определенного порога). Однако при этом за счет уменьшения времени вычислений сложность решаемых задач может быть увеличена (так, например, для учебной задачи суммирования может быть увеличен размер складываемого набора значений). Оценку получаемого при этом ускорения можно определить при помощи сформулированных закономерностей. Такая аналитическая оценка тем более полезна, поскольку решение таких более сложных вариантов задач на одном процессоре может оказаться достаточно трудоемким и даже невозможным, например, в силу нехватки оперативной памяти. С учетом указанных обстоятельств оценку ускорения, получаемую в соответствии с законом Густавсона-Барсиса, еще называют *ускорением масштабирования* (*scaled speedup*), поскольку данная характеристика может показать, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.