Интернет Университет Суперкомпьютерных технологий

Учебный курс

Основы параллельных вычислений

Лекция 4:

Анализ сложности вычислений и оценка возможности распараллеливания

Гергель В.П., профессор, д.т.н.
Нижегородский университет

Содержание

- Модель вычислений в виде графа "операции-операнды"
- Схема параллельного выполнения алгоритма
- Определение времени выполнения параллельного алгоритма
- Пример: Вычисление частных сумм последовательности числовых значений
- Пример: Умножение матриц
- Заключение

Введение

- Принципиальный момент при разработке параллельных алгоритмов - **анализ эффективности использования параллелизма**:
  - Оценка эффективности распараллеливания конкретных выбранных методов выполнения вычислений,
  - *Оценка максимально возможного ускорения процесса решения рассматриваемой задачи (анализ всех возможных способов выполнения вычислений)*

Граф "операции-операнды"…

- Модель в виде графа "операции-операнды" используется для описания существующих информационных зависимостей в выбираемых алгоритмах
- В наиболее простом виде модель основывается на предположениях:
  - время выполнения любых вычислительных операций является одинаковым и равняется 1,
  - передача данных между вычислительными устройствами выполняется мгновенно без каких-либо затрат времени.

Граф "операции-операнды"…

Множество операций, выполняемые в исследуемом алгоритме решения вычислительной задачи, и существующие между операциями информационные зависимости могут быть представлены в виде *ациклического ориентированного графа*

$$G = (V, R)$$

$V = \{1,\ldots,|V|\}$ – множество вершин графа, представляющих выполняемые операции алгоритма,

R – множество дуг графа; дуга *r(i,j)* принадлежит графу только если операция *j* использует результат выполнения операции *i*

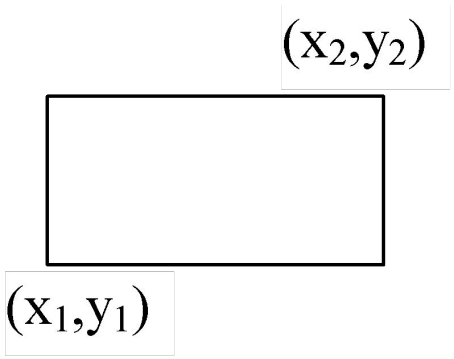
Вершины без входных дуг могут использоваться для задания операций ввода, а вершины без выходных дуг - для операций вывода.

$\overline{V}$ – множество вершин графа без вершин ввода,

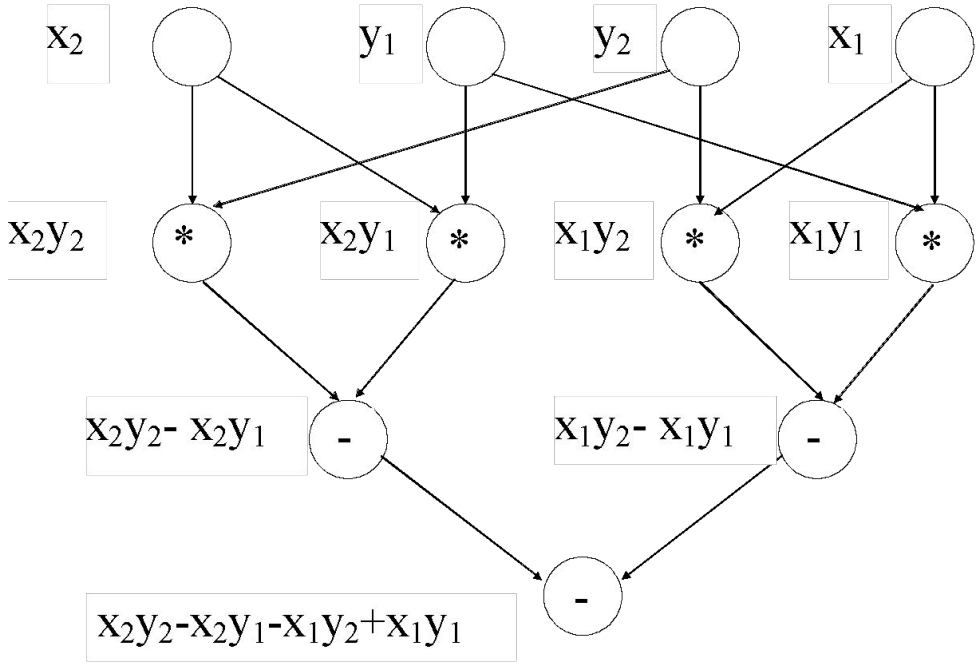
$d(G)$ – диаметр графа (длина максимального пути)

Граф "операции-операнды"…

Пример: граф алгоритма вычисления площади прямоугольника, заданного координатами двух противолежащих углов



$$S = (x_2-x_1)(y_2-y_1) = x_2y_2-x_2y_1-x_1y_2+x_1y_1$$



Граф "операции-операнды"

- Схемы вычислений обладают различными возможностями для распараллеливания, при построении модели вычислений может быть поставлена **задача выбора наиболее подходящей** для параллельного исполнения вычислительной схемы алгоритма
- Операции алгоритма, между которыми нет пути в рамках выбранной схемы вычислений, могут быть выполнены **параллельно**

Схема параллельного выполнения алгоритма

- Пусть ***p*** есть количество процессоров, используемых для выполнения алгоритма. Тогда для параллельного выполнения вычислений необходимо задать множество (*расписание*):

$$H_p = \{(i, P_i, t_i) : i \in V\}$$

  - ***i*** - есть номер операции,
  - P_i - есть номер процессора,
  - t_i - есть время начала выполнения *i*-ой операции.
- Должны выполняться условия:
  - один и тот же процессор не должен назначаться разным операциям в один и тот же момент времени:

$$\forall i, j \in V : t_i = t_j \Rightarrow P_i \neq P_j$$

  - к назначаемому моменту выполнения операции все необходимые данные уже должны быть вычислены:

$$\forall (i, j) \in R \Rightarrow t_j \geq t_i + 1$$

Определение времени выполнения параллельного алгоритма…

- Модель параллельного алгоритма:

$$A_p(G, H_p)$$

- Время выполнения параллельного алгоритма с заданным расписанием:

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

- Время выполнения параллельного алгоритма с оптимальным расписанием:

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

Определение времени выполнения параллельного алгоритма…

- Минимально возможное время решения задачи при заданном количестве процессоров (определение *наилучшей вычислительной схемы*):

$$T_p = \min_G T_p(G)$$

- Оценка наиболее быстрого исполнения алгоритма (при использовании *паракомпьютера* – системы с неограниченным числом процессоров):

$$T_\infty = \min_{p \geq 1} T_p$$

Определение времени выполнения параллельного алгоритма…

- Время выполнения последовательного алгоритма для заданной вычислительной схемы:

$$T_1(G) = \left|\overline{V}\right|$$

- Время выполнения последовательного алгоритма:

$$T_1 = \min_G T_1(G)$$

- Время последовательного решения задачи:

$$T_1^* = \min T_1$$

Подобные оценки необходимы для определения эффекта использования параллелизма

Определение времени выполнения параллельного алгоритма…

- **Теорема 1**

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

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

Определение времени выполнения параллельного алгоритма…

- **Теорема 2**

  Пусть для некоторой вершины вывода в вычислительной схеме алгоритма существует путь из каждой вершины ввода. Кроме того, пусть входная степень вершин схемы (количество входящих дуг) не превышает 2. Тогда минимально возможное время выполнения параллельного алгоритма ограничено снизу значением:

$$T_\infty(G) = \log_2 n,$$

  где ***n*** есть количество вершин ввода в схеме алгоритма

Определение времени выполнения параллельного алгоритма…

- **Теорема 3**

  При уменьшении числа используемых процессоров время выполнения алгоритма увеличивается пропорционально величине уменьшения количества процессоров, т.е. :

$$\forall q = cp, \quad 0 < c < 1 \Rightarrow T_p \le cT_q$$

Определение времени выполнения параллельного алгоритма…

- **Теорема 4**

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

$$\forall p \Rightarrow T_p < T_\infty + T_1 / p$$

Определение времени выполнения параллельного алгоритма…

Теорема 5

Времени выполнения алгоритма, которое сопоставимо с минимально возможным временем T_∞, можно достичь при количестве процессоров порядка $p \sim T_1/T_\infty$, а именно:

$$p \geq T_1 / T_\infty \Rightarrow T_p \leq 2T_\infty$$

При меньшем количестве процессоров время выполнения алгоритма не может превышать более, чем в 2 раза, наилучшее время вычислений при имеющемся числе процессоров, т.е.:

$$p < T_1 / T_\infty \Rightarrow \frac{T_1}{p} \leq T_p \leq 2\frac{T_1}{p}$$

Определение времени выполнения параллельного алгоритма…

- **Рекомендации**
  - при выборе вычислительной схемы алгоритма должен использоваться граф с минимально возможным диаметром (теорема 1),
  - для параллельного выполнения целесообразное количество процессоров определяется величиной $p \sim T_1/T_\infty$ (теорема 5),
  - время выполнения параллельного алгоритма ограничивается сверху величинами, приведенными в теоремах 4 и 5.

Пример: Вычисление частных сумм…

- Задача нахождения частных сумм последовательности числовых значений (*prefix sum problem*):

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

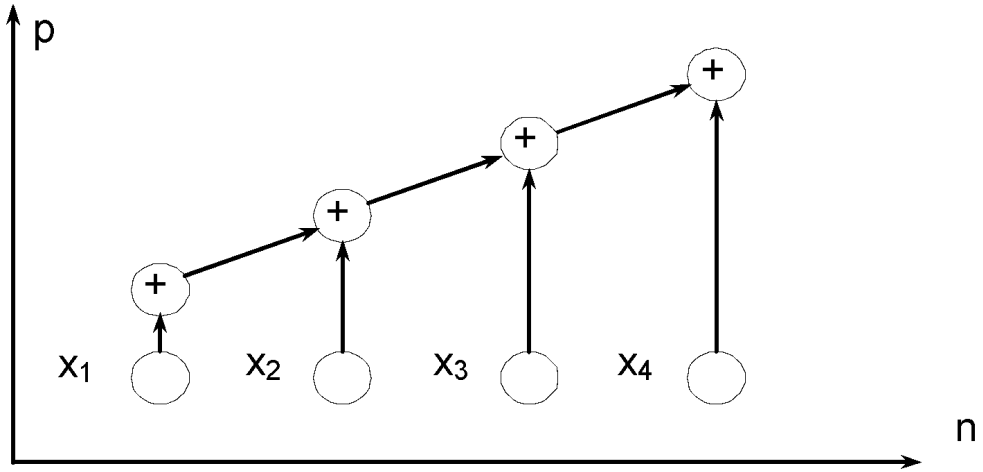
- Задача вычисления общей суммы имеющегося набора значений:

$$S = \sum_{i=1}^{n} x_i$$

Пример: Вычисление частных сумм…

- Последовательный алгоритм суммирования элементов числового вектора

$$S = \sum_{i=1}^{n} x_i, \qquad G_1 = (V_1, R_1)$$

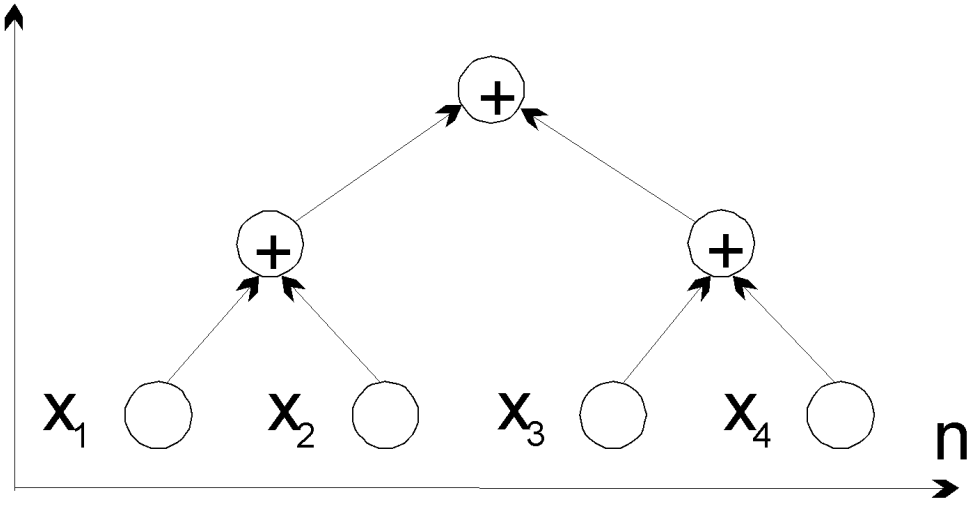


*Данный "стандартный" алгоритм суммирования допускает только **строго последовательное исполнение** и не может быть распараллелен*

Пример: Вычисление частных сумм…

- **Каскадная схема суммирования**

$$G_2 = (V_2, R_2)$$



- V_2 = { ($v_{i1},\ldots,v_{ili}$), $0 \le i \le k$, $1 \le l_i \le 2^{-i}n$ } есть вершины графа,
- ($v_{01},\ldots, v_{0n}$) есть операции ввода,
- ($v_{11},\ldots,v_{1n/2}$) есть операции первой итерации и т.д.,
- R_2 = { ($v_{i-1,2j-1}v_{ij}$),($v_{i-1,2j}v_{ij}$), $1 \le i \le k$, $1 \le l_i \le 2^{-i}n$ } есть множество дуг графа.

Пример: Вычисление частных сумм…

- Количество итераций каскадной схемы суммирования:

$$k = \log_2 n$$

- Общее количество операций суммирования:

$$K_{посл} = n/2 + n/4 + \ldots + 1 = n - 1$$

- При параллельном исполнении отдельных итераций каскадной схемы общее количество параллельных операций суммирования является равным:

$$K_{пар} = \log_2 n$$

Пример: Вычисление частных сумм…

- Показатели ускорения и эффективности каскадной схемы алгоритма суммирования:

$$S_p = T_1 / T_p = (n-1)/\log_2 n,$$
$$E_p = T_1 / pT_p = (n-1)/(p\log_2 n) = (n-1)/((n/2)\log_2 n),$$

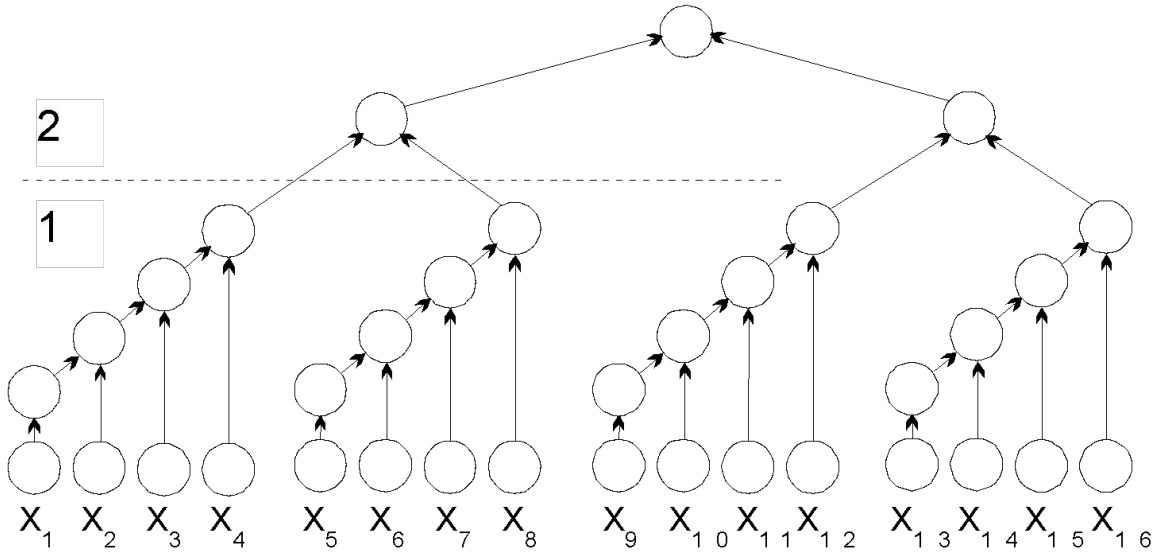
где *p=n/2* есть необходимое для выполнения каскадной схемы количество процессоров.

  - время параллельного выполнения каскадной схемы совпадает с оценкой для паракомпьютера (теорема 2),
  - эффективность использования процессоров уменьшается при увеличении количества суммируемых значений:

$$\lim E_p = 0 \quad при \quad n \to \infty$$

Пример: Вычисление частных сумм…

- **Модифицированная каскадная схема**:
  - Все суммируемые значения подразделяются на $(n/\log_2 n)$ групп, в каждой из которых содержится $(\log_2 n)$ элементов; для каждой группы вычисляется сумма значений при помощи последовательного алгоритма суммирования;
  - На втором этапе для полученных $(n/\log_2 n)$ сумм отдельных групп применяется обычная каскадная схема:



Пример: Вычисление частных сумм…

- Для выполнения первого этапа требуется $(\log_2 n)$ выполнение параллельных операций при использовании $p = (n/\log_2 n)$ процессоров
- Для выполнения второго этапа необходимо $\log_2(n/\log_2 n) \le \log_2 n$ параллельных операций для $p=(n/\log_2 n)/2$ процессоров
- Время выполнения параллельного алгоритма составляет

$$T_P = 2\log_2 n$$

для $p = (n/\log_2 n)$ процессоров.

Пример: Вычисление частных сумм…

- С учетом полученных оценок показатели ускорения и эффективности модифицированной каскадной схемы определяются соотношениями:

$$S_p = T_1 / T_p = (n-1)/2\log_2 n,$$

$$E_p = T_1 / pT_p = (n-1)/(2(n/\log_2 n)\log_2 n) = (n-1)/2n$$

  - По сравнению с обычной каскадной схемой ускорение уменьшилось в 2 раза,
  - Для эффективности нового метода суммирования можно получить асимптотически ненулевую оценку снизу:

$$E_p = (n-1)/2n \geq 0.25,\ \lim E_p = 0.5\ \textit{при}\ n \to \infty.$$

  - Модифицированный каскадный алгоритм является стоимостно-оптимальным (стоимость вычислений пропорциональна времени выполнения последовательного алгоритма):

$$C_p = pT_p = (n/\log_2 n)(2\log_2 n) = 2n$$

Пример: Вычисление частных сумм…

- **Вычисление всех частных сумм…**
  - Вычисление всех частных сумм на скалярном компьютере может быть получено при помощи обычного последовательного алгоритма суммирования при том же количестве операций

$$T_1 = n$$

  - При параллельном исполнении применение каскадной схемы в явном виде не приводит к желаемым результатам.

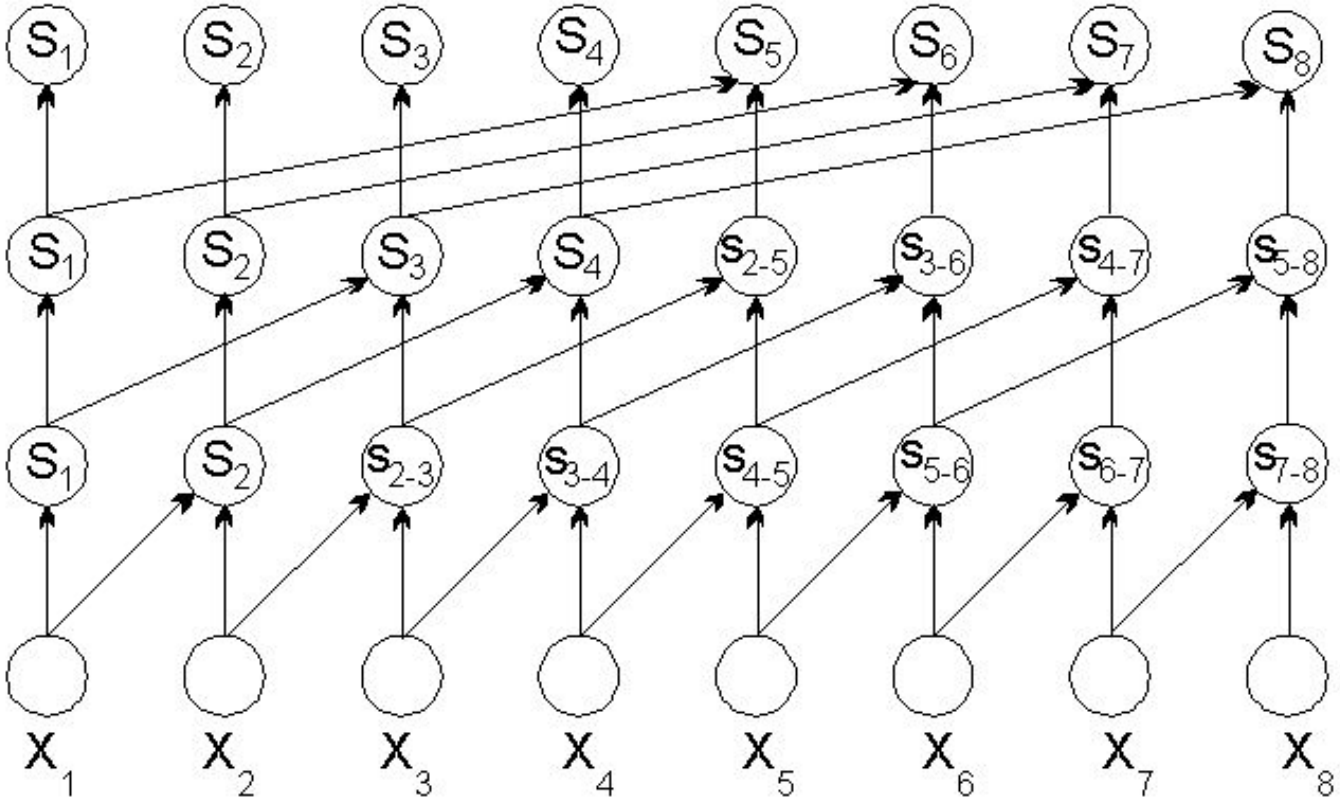
Достижение эффективного распараллеливания требует привлечения новых подходов (может быть, даже не имеющих аналогов при последовательном программировании) для разработки новых параллельно-ориентированных алгоритмов решения задач

Пример: Вычисление частных сумм…

- **Вычисление всех частных сумм…**
  - Алгоритм, обеспечивающий получение результатов за $log_2 n$ параллельных операций:
    - Перед началом вычислений создается копия S вектора суммируемых значений ($S=x$),
    - Далее на каждой итерации суммирования i, $1 \le i \le log_2 n$, формируется вспомогательный вектор Q путем сдвига вправо вектора S на 2^{i-1} позиций (освобождающиеся при сдвиге позиции слева устанавливаются в нулевые значения); итерация алгоритма завершается параллельной операцией суммирования векторов S и Q.

Пример: Вычисление частных сумм…

- Вычисление всех частных сумм…



Пример: Вычисление частных сумм

- **Вычисление всех частных сумм**
  - Общее количество выполняемых алгоритмом скалярных операций определяется величиной:

$$\mathrm{K}_{посл} = n \log_2 n$$

  - Необходимое количество процессоров определяется количеством суммируемых значений:

$$p = n$$

  - Показатели ускорения и эффективности:

$$S_p = T_1 / T_p = n / \log_2 n$$

$$E_p = T_1 / pT_p = n / \left(p \log_2 n \right) = n / n \log_2 n = 1 / \log_2 n$$

Пример: Умножение матриц…

Умножение матриц:

$$C = A \cdot B$$

ИЛИ

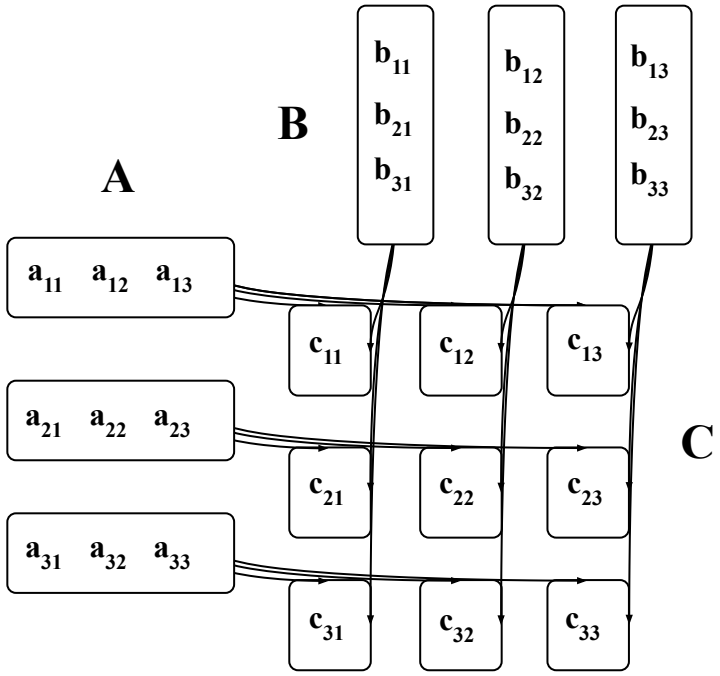
$$\begin{pmatrix} c_{0,0}, & c_{0,1}, & \ldots, & c_{0,l-1} \\ & & \ldots & \\ c_{m-1,0}, & c_{m-1,1}, & \ldots, & c_{m-1,l-1} \end{pmatrix} = \begin{pmatrix} a_{0,0}, & a_{0,1}, & \ldots, & a_{0,n-1} \\ & & \ldots & \\ a_{m-1,0}, & a_{m-1,1}, & \ldots, & a_{m-1,n-1} \end{pmatrix} \begin{pmatrix} b_{0,0}, & b_{0,1}, & \ldots, & a_{0,l-1} \\ & & \ldots & \\ b_{n-1,0}, & b_{n-1,1}, & \ldots, & b_{n-1,l-1} \end{pmatrix}$$

ИЛИ

$$c_{ij} = \left(a_i, b_j^T\right) = \sum_{k=0}^{n-1} a_{ik} \cdot b_{kj}, 0 \le i < m, 0 \le j < l$$

Пример: Умножение матриц

Граф информационных зависимостей (до уровня операций умножения строк матрицы *A* и столбцов матрицы *B*):



- Задача умножения матрицы на вектор может быть сведена к выполнению ***m·n*** независимых операций умножения строк матрицы ***A*** на столбцы матрицы ***B***

Заключение

- Описывается модель вычислений в виде графа "операции-операнды", которая может использоваться для описания существующих информационных зависимостей в выбираемых алгоритмах решения задач
- Приводятся теоретические оценки, которые могут быть использованы при определении максимального возможного распараллеливания
- Для демонстрации применимости рассмотренных моделей и методов анализа параллельных алгоритмов в разделе рассматриваются задачи нахождения частных сумм последовательности числовых значений и умножения матриц

Вопросы для обсуждения

- Как определяется время выполнения параллельного алгоритма?
- Как определить минимально возможное время решения задачи?
- Какие оценки следует использовать в качестве характеристики времени последовательного решения задачи?
- Как определить минимально возможное время параллельного решения задачи по графу "операнды – операции"?
- При каком числе процессоров могут быть получены времена выполнения параллельного алгоритма, сопоставимые по порядку с оценками минимально возможного времени решения задачи?

Темы заданий для самостоятельной работы

Разработайте модель и выполните оценку максимально возможных показателей ускорения и эффективности параллельных вычислений:

1. Для задачи скалярного произведения двух векторов,
2. Для задачи поиска максимального и минимального значений для заданного набора числовых данных,
3. Для задачи нахождения среднего значения для заданного набора числовых данных,
4. Для задачи умножения матрицы на вектор,
5. Для задачи матричного умножения

Литература…

- **Гергель В.П.** Теория и практика параллельных вычислений. - М.: Интернет-Университет, БИНОМ. Лаборатория знаний, 2007. – Лекция 2
- **Воеводин** В.В., Воеводин Вл.В. Параллельные вычисления. – СПб.: БХВ-Петербург, 2002.

Дополнительные учебные курсы:

- **Барский А.Б.** Параллельное программирование. — http://www.intuit.ru/department/se/parallprog/
- **Воеводин В.В.** Вычислительная математика и структура алгоритмов. — http://www.intuit.ru/department/calculate/calcalgo/

Следующая тема

- **Общая схема разработки параллельных методов**

Контакты

Гергель В.П., профессор, д.т.н., декан факультета вычислительной математики и кибернетики

Нижегородский университет

gergel@unn.ru

http://www.software.unn.ru/?dir=17