Основные подходы к распараллеливанию

Судаков А.А.

"Параллельные и распределенные вычисления" Лекция 16

План

- Конвейер
- Матричная обработка
- Распараллеливание циклов

Конвейер

- Конвейерная обработка – метод распараллеливания существенно последовательных операций
- Процессоры соединяются так, чтобы результат работы одного процессора поступал на вход другого (линейная топология)
- Сложная операция разбивается на несколько последовательных стадий, каждая стадия выполняется своим процессором

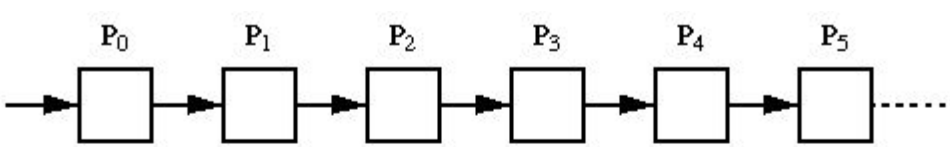


Figure 5.1 Pipelined processes

Использование конвейера

- Параллелизм на уровне инструкций
  - Конвейерная обработка в процессорах
- Параллелизм на уровне процедур
  - Конвейерное объединение потоков
- Параллелизм на уровне приложений
  - Передача выхода одной программы на вход другой
  - Передача сообщений

Когда конвейер эффективный

- Три случая

1. С помощью конвейера необходимо решить несколько последовательных задач
2. С помощью конвейера необходимо решить одну задачу для большой последовательности входных данных
3. Когда каждый процессор еще до завершения своей работы может отправить данные следующему процессору и запустить его работу

Пример: конвейер первого типа

- Пример: Нахождение суммы нескольких значений
  - Есть p процессоров
  - У каждого процессора есть свои данные d_i
  - Необходимо найти сумму $\sum_i d_i$

Реализация

```
// i – номер поточного процессора
// d- дані поточного процесора
// sum – проміжне значення, яке на
// останньому процесорі буде містити
// результат розв’язання задачі
recv(i-1,&sum)
sum=+d
send(i+1,sum)
```

Время выполнения первой задачи

- Каждый процессор получает результат предыдущего
- Добавляет к нему свое значение и отправляет результат дальше
- Время вычисления первой суммы:
  - Суммарное время работы всех процессоров
  - Время передачи данных от первого процессора до последнего

$$(p-1)t_c + (p-1)dt_o + pdt$$

- Это время называется временем заполнения конвейера (существенно последовательная операция)
- Время выполнения последовательного алгоритма
- Ускорение
- Для решения одной задачи конвейер не эффективен!

$$T_1 = pdt$$

$$\sim \frac{1}{t_c / t + t_o / t + 1} < 1$$

Время решения нескольких задач

- Пусть задачу необходимо решить m раз для m наборов данных на каждом процессоре
- Каждую следующую операцию процессор будет выполнять за время

$$dt_{next} = \max(t, t_o)$$

- Общее время решения задачи

$$T_p = (p-1)t_c + (p-1)dt_o + pdt + d(m-1)t_{next}$$

- Время решения m задач на 1 процессоре

$$T_1 = mpdt$$

- Ускорение

$$k_n \sim \frac{pmdt}{pt_c + pdt + dt_o + d(m-1)t_{next}} \sim \frac{pmt}{pt_o + pt + (m-1)t_{next}}$$

Время выполнения при большом количестве задач

- Предел коэффициента ускорения при m->∞

$$k_n \sim \frac{pt}{t_{next}}$$

- При очень большом количестве задач ускорение стремится к идеальному
- При малом количестве ограничивается законом Амдала

Иллюстрация

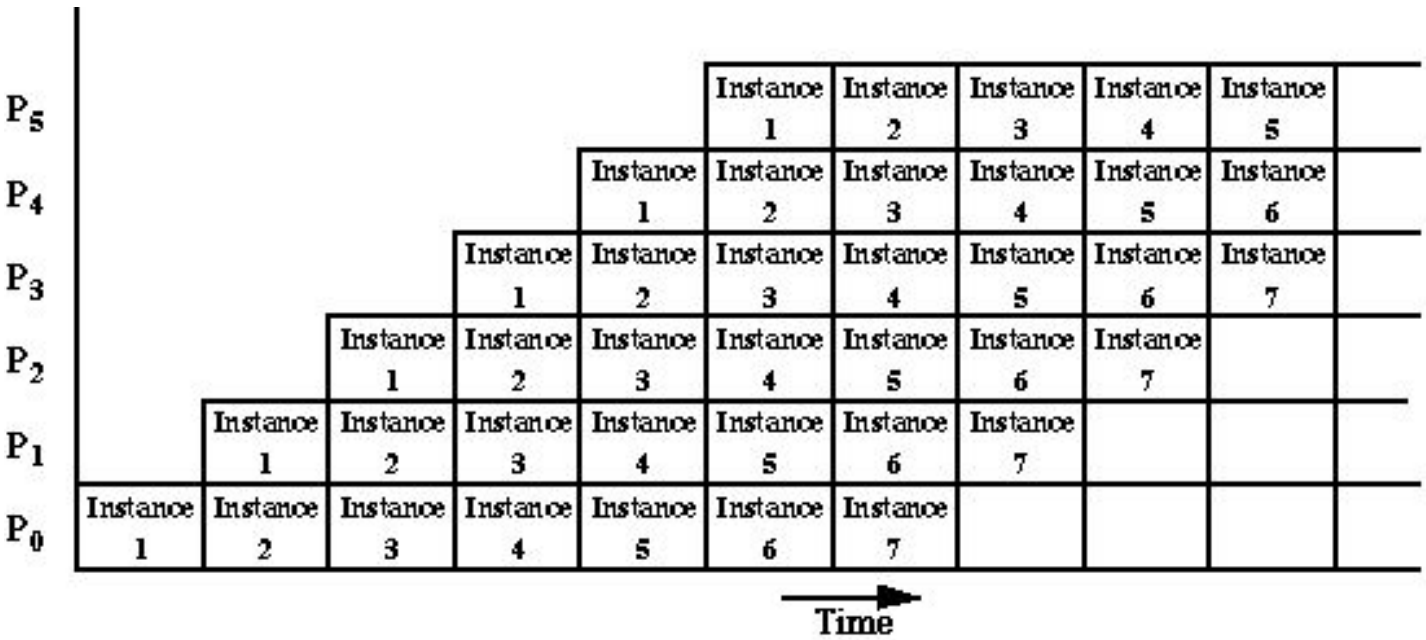
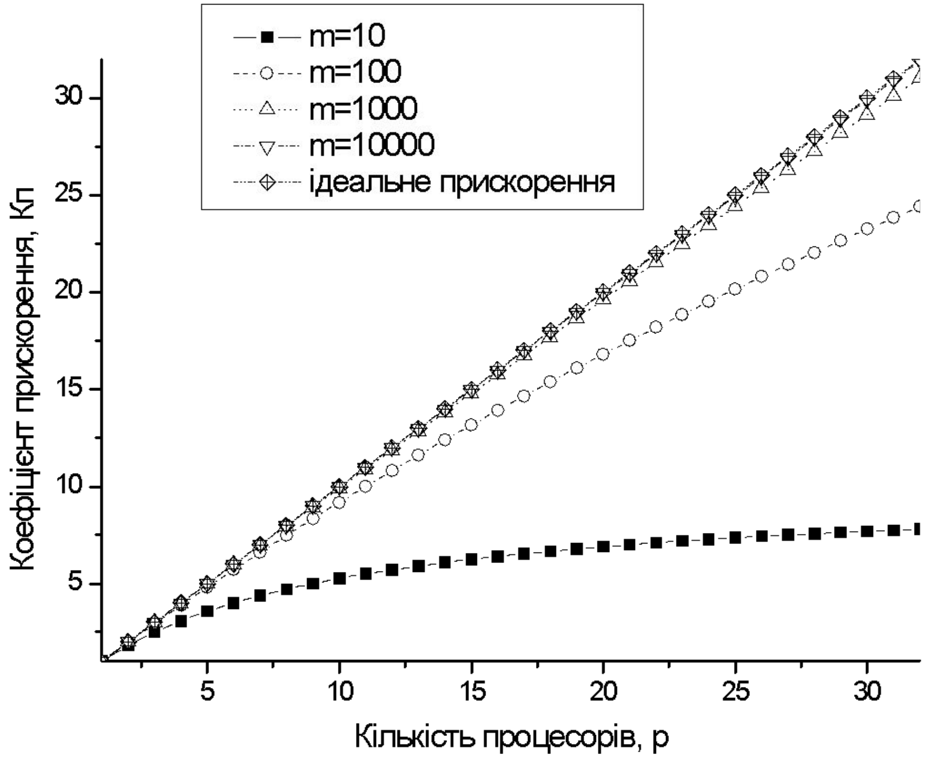


Figure 5.2 Space-time diagram of a pipeline

Графическая иллюстрация

- Доля последовательных вычислений зависит от количества задач



Вводы относительно конвейера первого типа

- Эффективность конвейера всегда меньше единицы
- Для получения высоких коэффициентов ускорения необходимо, чтобы скорость обмена дынными была по возможности меньше
- Коэффициент ускорения асимптотически стремится к значению $$k_n \sim \frac{pt}{t_o}$$
  - Эффективность конвейера растет при увеличении времени, которое тратит процессор на обработку данных по сравнению с временем передачи
  - Даже для медленных каналов связи при большом количестве операций алгоритма конвейер может давать ускорение
- Для эффективной работы конвейера его необходимо сильно загрузить

Конвейер второго типа

- Пример: сортировка массива значений
- На входе есть большой массив данных
- Каждый процессор получает информацию от предыдущего и передает наибольшее из своих значений следующему процессору
- В результате процессор с меньшим номером будет содержать элементы массива с меньшими значениями

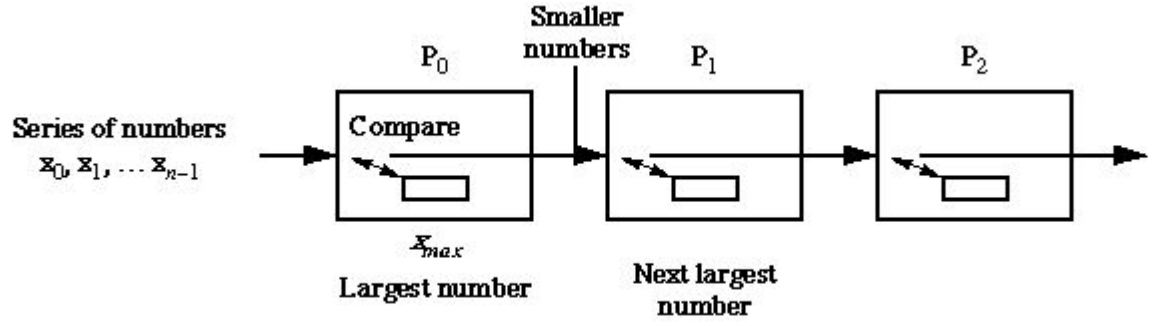


Figure 5.7 Pipeline for sorting using insertion sort

Программа

```
recv(i-1,x)
for(j=0;j<(n-i); j++){
   recv(i-1,number)
   if(number > x) {
   Send(i+1,x);
} else {
   send(i+1, number);
}
}
```

Оценка времени

- Каждый процессор выполняет порядка n операций сравнения параллельно с остальными
- Самый быстрый последовательный алгоритм требует порядка $n(\log_2 n)$ операций
- Коэффициент ускорения
- Эффективность достаточно маленькая

$$k_n = \frac{T_1}{T_p} = \frac{O(n\log_2 n)}{O(n)} = O(\log_2 n)$$

Конвейер третьего типа

- Одновременная передача данных и обработка (Send-ahead)
- Если полученные от предыдущего процессора данные можно передать на следующий процессор, а потом начать их обработку

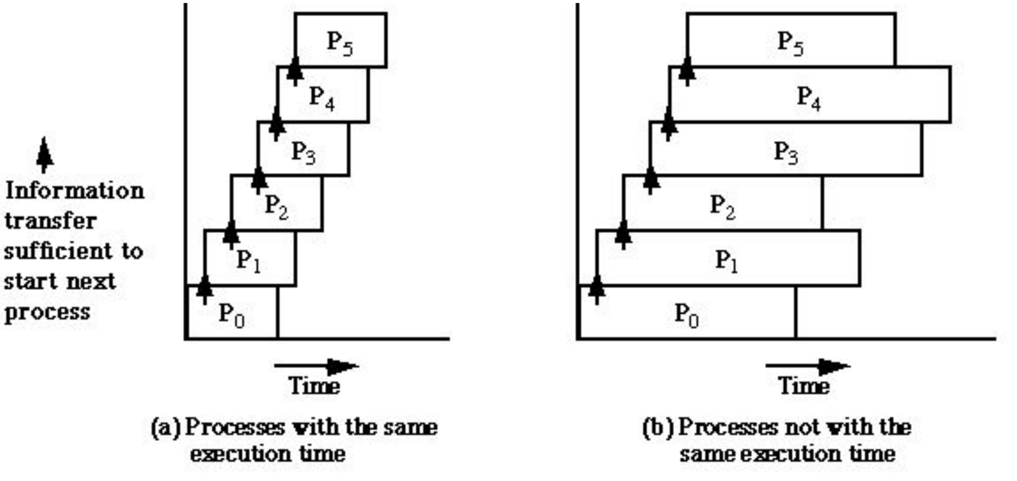


Figure 5.5 Pipeline processing where information passes to next stage before end of process

Пример конвейера третьего типа

- Обратный ход метода Гаусса
- Матрица треугольной формы

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_3 = b_n$$

$$\ldots..$$

$$\ldots..$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

$$a_{11}x_1 = b_1$$

Решение

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_3 = b_n$$

.....

.....

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3$$

$$a_{21}x_1 + a_{22}x_2 = b_2$$

$$a_{11}x_1 = b_1$$

Из последнего уравнения $x_1 = \dfrac{b_1}{a_{11}}$

Из предпоследнего $x_2 = \dfrac{b_2 - a_{21}x_2}{a_{22}}$

Для любого уравнения $x_n = \dfrac{b_n - \sum_{i=1}^{n-1} a_{ni}x_i}{a_{nn}}$

Последовательная программа

```
// по всем неизвестным
for(i=1; i<=n; i++){
   sum=0;
// по всем найденным неизвестным
   for(j=1;j<i; j++){
    sum+=a[i][j]*x[j];
    x[i]=(b[i]-sum)/a[i][j];
}
}
```

Параллельная программа

- Процессор 1 –вычисляет x_1 и передает его дальше
- Процессор 2 принимает x_1 передает его дальше, вычисляет x_2 и тоже передает его дальше
- Процессор 3 принимает x_1 , принимает x_2 , передает их дальше, вычисляет x_3 и передает его дальше
- …
- Вычисление выполняется параллельно с передачей данных
- разные процессоры работают одновременно

Параллельная программа

```
for(j=1;j<=i; j++){
    recv(p-1,x[j])
    send(p+1,x[j])
}
sum=0;
for(j=1; i<=j; j++){
     sum+=a[i][j]*x[j];
}
    x[i]=(b[i]-sum)/a[i][j];
    send(p+1,x[i])
```

Эффективность

- Последовательный алгоритм O(n^2) операций
- Параллельный алгоритм – каждый процессор порядка O(n) операций
- Ускорение порядка O(n)
- С увеличением порядка матрицы ускорение растет

Выводы относительно конвейера

- Самый простой и дешевый вариант распараллеливания
- Возможность распараллеливания принципиально последовательных операций
- Возможность одновременного выполнения передачи данных и их обработки (асинхронные операции)

- Эффективность всегда меньше 1

Матричная (векторная, параллельная) обработка

- Каждый процессор получает часть своей задачи и выполняет ее параллельно с другими процессорами
- В конце выполнения процессоры синхронизируются (барьер)

Ускорение

- Ускорение при равномерной загрузке процессоров стремится к количеству процессоров
- Эффективность стремится к единице
- Процессоры большую часть времени загружены

Преимущества

- Высокая эффективность
- Обмен мало влияет на скорость выполнения

Недостатки

- Синхронный режим работы
  - Во время синхронизации процессоры простаивают
- Нет возможности выполнять обмен одновременно с вычислениями
- Требует большого количества одинаковых процессоров
- Обычно дорогостоящее решение

Векторно-конвейерная схема

- Вектор – массив элементов одинакового типа
- Векторные операции – большое количество одинаковых операций с разными данными

  x[i]+y[i]=z[i]
- Конвейер второго типа может быть эффективным для таких операций

сложение двух чисел

- Сложение чисел стандарт ANSI/IEEE

[A:] сравнение порядков и определение меньшего числа

[B:] сдвиг мантиссы числа с меньшим порядком, чтобы порядки стали одинаковыми

[C:] складываются мантиссы полученных чисел

[D:] результат нормализируется

[E:] проверка на возникновение исключительных ситуаций

[F:] Округление

Пример сложения

- x+y, где x=1234,00 y = -567,8

Крок	x	y	$s = x + y$
[A:]	0.1234**E**4	**-0.5678**E**3**	
[B:]	0.12340**E**4	-0.05678**E**4	
[C:]			0.066620**E**4
[D:]			0.66620**E**3
[E:]			0.66620**E**3
[F:]			0.6662**E**3

n

Оценка времени

- Время выполнения одной стадии t
- Время сложения двух чисел 6t
- Сложение двух векторов длины n выполнится за 6tn

Диаграмма состояний

Час	t	$2t$	$3t$	$4t$	$5t$	$6t$	$7t$	$8t$
Крок								
[A:]	$x_1 + y_1$						$x_2 + y_2$	
[B:]		$x_1 + y_1$						$x_2 + y_2$
[C:]			$x_1 + y_1$					
[D:]				$x_1 + y_1$				
[E:]					$x_1 + y_1$			
[F]:						$x_1 + y_1$		

Конвейерное выполнение

- После выполнения первой стадии с первым числом сразу же запускается первая стадия со вторым числом

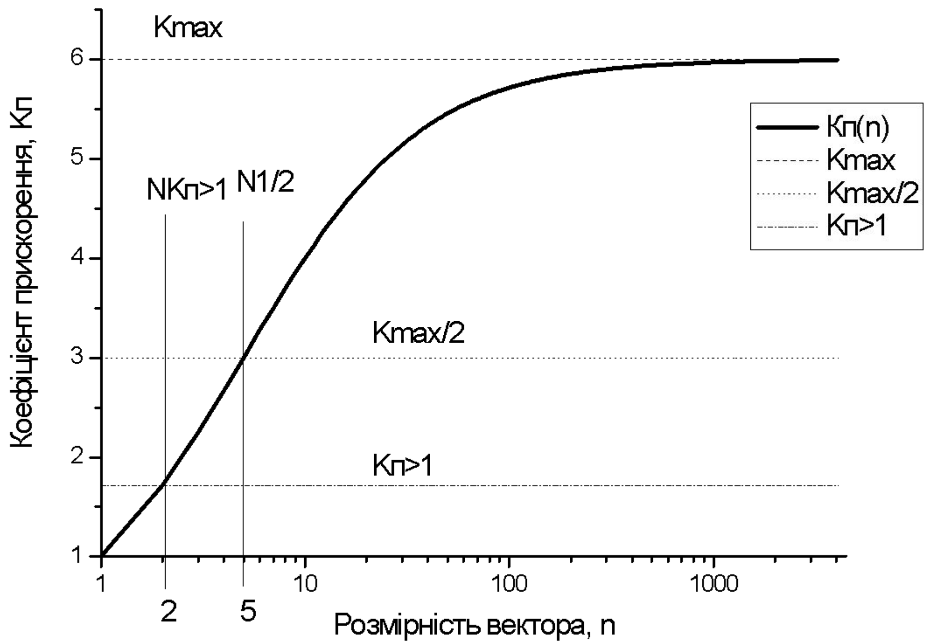
Час	t	$2t$	$3t$	$4t$	$5t$	$6t$	$7t$	$8t$
Крок								
[A:]	$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$	$x_4 + y_4$	$x_5 + y_5$	$x_6 + y_6$	$x_7 + y_7$	$x_8 + y_8$
[B:]		$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$	$x_4 + y_4$	$x_5 + y_5$	$x_6 + y_6$	$x_7 + y_7$
[C:]			$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$	$x_4 + y_4$	$x_5 + y_5$	$x_6 + y_6$
[D:]				$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$	$x_4 + y_4$	$x_5 + y_5$
[E:]					$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$	$x_4 + y_4$
[F]:						$x_1 + y_1$	$x_2 + y_2$	$x_3 + y_3$

Оценка времени

- Время сложения двух векторов
- Коэффициент ускорения

$$6t + (n-1)t = (n+5)t$$

$$k_n = \frac{6nt}{(n+5)t} = \frac{6n}{(n+5)}$$



Параметры векторно-конвейерной системы

- Размерность вектора, при котором скорость работы векторного процессора становится большей, чем скалярного (2 в данном случае) $N_{k_n > 1}$
- Максимально возможный коэффициент ускорения (векторная скорость света) = 6 $k_{n\max}$
  - При размерности вектора 4096 ускорение равно 5.99268
- Размерность вектора при которой производительность равна половине от максимальной (5 в данном случае) $N_{1/2}$

Распараллеливание циклов

- Циклы типа for() обычно хорошо распараллеливаются
- Циклы типа while распараллеливаются обычно плохо
- Два подхода
  - Развертка циклов (unroll)
    - Векторизация циклов
  - Разбивка на блоки (blocking)

Разбивка на блоки

```
for(i=0; i<N;i++){
   // тіло циклу
   a[i]=b[i]+1;
}
```

- Цикл разбивается на блоки
- каждый блок выполняется своим процессором с помощью параллельной схемы

Пример разбивки на блоки

- Цикл разбивается на p циклов
- Каждый процессор выполняет свой цикл

```
for(j=0;j<p; j++){
for(i=j; i<N;i+=p){
    // тіло циклу
    a[i]=b[i]+1
}
}
```

Внутренний цикл выполняется своим процессором

Эффективность

- При отсутствии связи между данными внутри цикла эффективность стремится к 1

Развертка циклов

- Часть операций цикла можно заменить последовательностью операций и выполнить с помощью конвейера

```
for(i=0; i<N;i+=k){
    // тіло циклу
    a[i]=b[i]+1;
    a[i+1]=b[i+1]+1;
    a[i+2]=b[i+2]+1;
    a[i+3]=b[i+3]+1;
…
    a[i+k-2]=b[i+k-2]+1;
    a[i+k-1]=b[i+k-1]+1;
}
```

Ускорение почти в k раз при большом k

Векторизация циклов

- Часть операций цикла можно заменить последовательностью операций и выполнить с помощью векторных команд (mmx, sse)

for(i=0; i<N;i+=k){

// a и b рассматриваются как векторы размерности k

a[начиная с i]=b[начиная с i]+1;

}

- Ускорение почти в k раз

Работа с большими матрицами

- В прикладных задачах часто возникает проблема работы с матрицами большого размера (100000x100000)
- Для размещения такой матрицы чисел типа double потребуется 76 ГБайт
- В оперативную память одной машины такая матрица не влезет
- Используются блочные методы

Блочные методы

- Матрица разбивается на части – блоки (rxq блоков)
- Каждый блок хранится в памяти одного узла массивно параллельной системы (кластера)
- Говорят: «На матрицу накладывается процессорная сетка»
- Каждому узлу сетки (блоку матрицы) соответствует процессор с координатами (i,j)
- Для выполнения операций с матрицами процессоры должны обмениваться данными между собой (топология решетка)

Матрица

Математика блочных операций

- Математически операции с блочными матрицами выполняются так же, как и операции с обычными матрицами, но умножение чисел заменяется умножением матриц, сложение…

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1k} \\ & \dots & & \\ A_{k1} & A_{k2} & \dots & A_{kk} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1k} \\ & \dots & & \\ B_{k1} & B_{k2} & \dots & B_{kk} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \dots & C_{1k} \\ & \dots & & \\ C_{k1} & C_{k2} & \dots & C_{kk} \end{bmatrix}$$

Блок матриці C вычисляется по формуле

$$C_{ij} = \sum_{l=1}^{k} A_{il} B_{lj}$$

Распараллеливание блочных операций

- Вычисление каждого блока результата может выполняться параллельно с вычислением других блоков результата
- Эффективность таких операций увеличивается при увеличении размерности матриц
  - Количество операций обработки данных порядка $O(m^3)$
  - Время передачи данных $O(m^2)$

Геометрическое распараллеливание

- Процессоры, которые интенсивно взаимодействуют между собой должны находится «ближе»
    - например располагаться на одном узле многопроцессорной машины

Вопросы?