Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Образовательный комплекс

Введение в методы параллельного программирования

Раздел 11.

Алгоритмы на графах

Гергель В.П., профессор, д.т.н.
Кафедра математического обеспечения ЭВМ

Содержание

- Обработка графов
- Задача поиска всех кратчайших путей
- Задача нахождения минимального охватывающего дерева
- Задача оптимального разделения графов
- Заключение

Обработка графов

- Математические модели в виде графов широко используются при моделировании разнообразных явлений, процессов и систем
- Граф G есть пара:

$$G = (V, R)$$

V – набор вершин графа:

$$v_i, 1 \le i \le n$$

R – набор ребер графа:

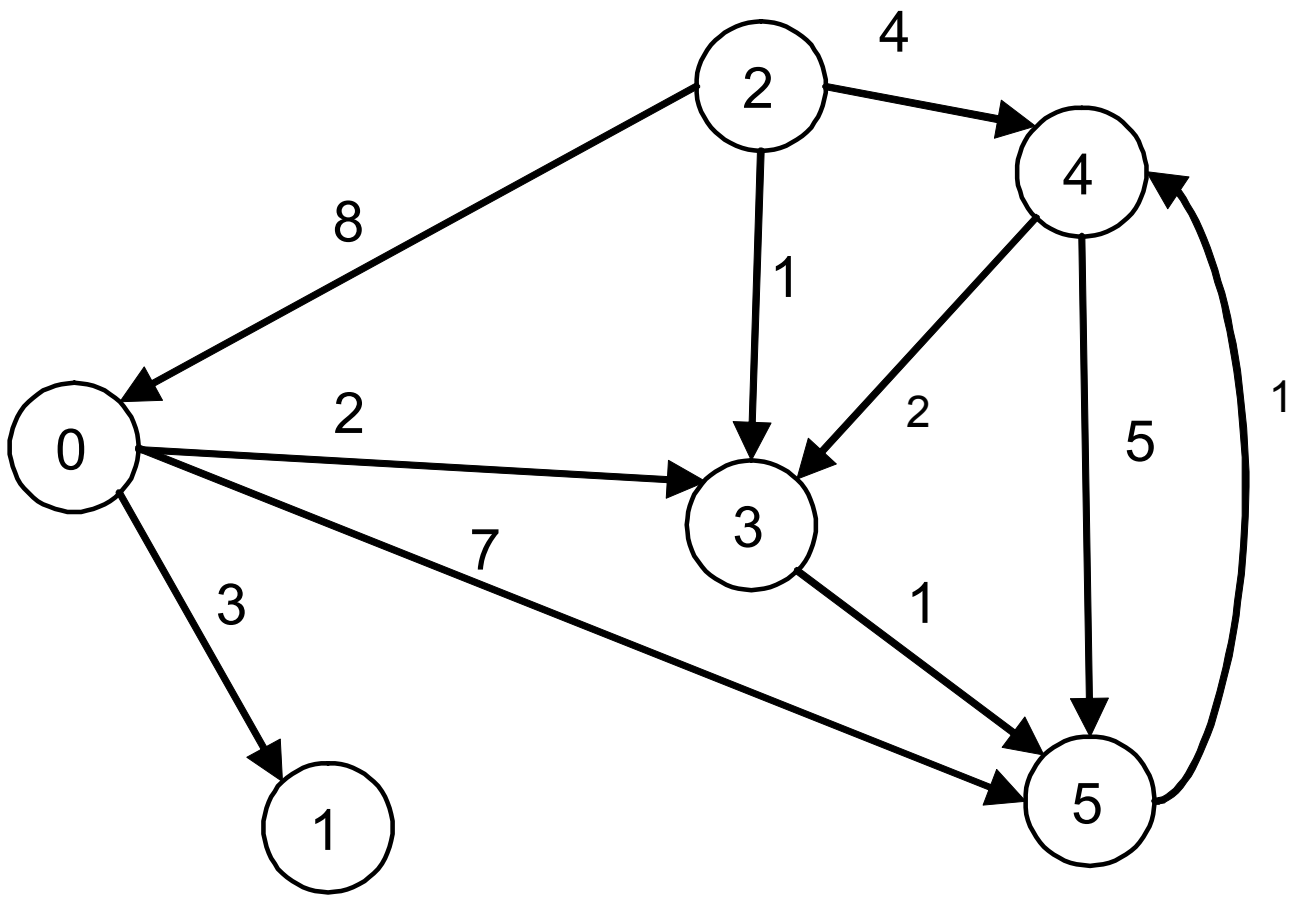
$$r_j = (v_{s_j}, v_{t_j}), 1 \le j \le m$$

- В общем случае дугам графа могут приписываться некоторые числовые характеристики (веса) :

$$w_j, 1 \le j \le m$$

Обработка графов

- Пример взвешенного ориентированного графа



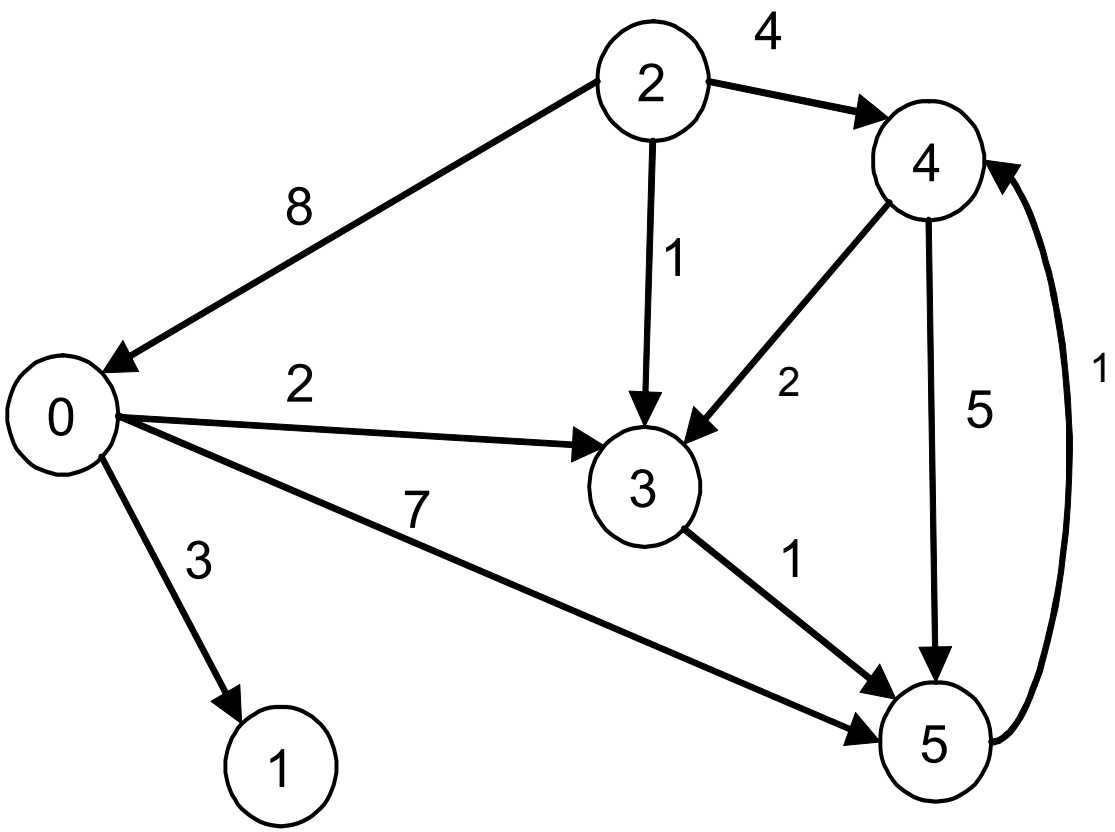
Обработка графов

- При малом количестве дуг в графе целесообразно использовать для определения графов списки, перечисляющие имеющиеся в графах дуги
- Представление достаточно плотных графов, для которых почти все вершины соединены между собой дугами, может быть эффективно обеспечено при помощи *матрицы смежности*

$$A = (a_{ij}), 1 \le i, j \le n, \quad a_{ij} = \begin{cases} w(v_i, v_j), & \text{если } (v_i, v_j) \in R, \\ 0, & \text{если } i = j, \\ \infty, & \text{иначе.} \end{cases}$$

Обработка графов

- Пример матрицы смежности



$$\begin{pmatrix} 0 & 3 & \infty & 2 & \infty & 7 \\ \infty & 0 & \infty & \infty & \infty & \infty \\ 8 & \infty & 0 & 1 & 4 & \infty \\ \infty & \infty & \infty & 0 & \infty & 1 \\ \infty & \infty & \infty & 2 & 0 & 5 \\ \infty & \infty & \infty & \infty & 1 & 0 \end{pmatrix}$$

Задача поиска всех кратчайших путей

- **Постановка задачи**:
  - Дан граф G, каждому ребру которого приписан неотрицательный вес
  - Граф полагаем ориентированным
  - Для имеющегося графа *G* требуется найти минимальные длины путей между каждой парой вершин графа
  - В качестве метода, решающего задачу поиска кратчайших путей между всеми парами пунктов назначения, далее используется *алгоритм Флойда* (*Floyd*)

Задача поиска всех кратчайших путей

- **Последовательный алгоритм Флойда**
  - Сложность алгоритма имеет порядок

$$O(n^3)$$

```
// Алгоритм 11.1
// Последовательный алгоритм Флойда
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      A[i,j] = min(A[i,j],A[i,k]+A[k,j]);
```

Задача поиска всех кратчайших путей

- **Разделение вычислений на независимые части**
  - Эффективный способ организации параллельных вычислений состоит в одновременном выполнении нескольких операций обновления значений матрицы *A*
  - На итерации *k* не происходит изменения элементов *Aik* и *Akj* ни для одной пары индексов *(i,j)*
  - Докажем это:

$$Aij \leftarrow \min(Aij, Aik + Akj)$$

    - *Для i=k получим:*

$$Akj \leftarrow min(Akj, Akk + Akj),$$

но тогда значение Akj не изменится, т.к. Akk=0

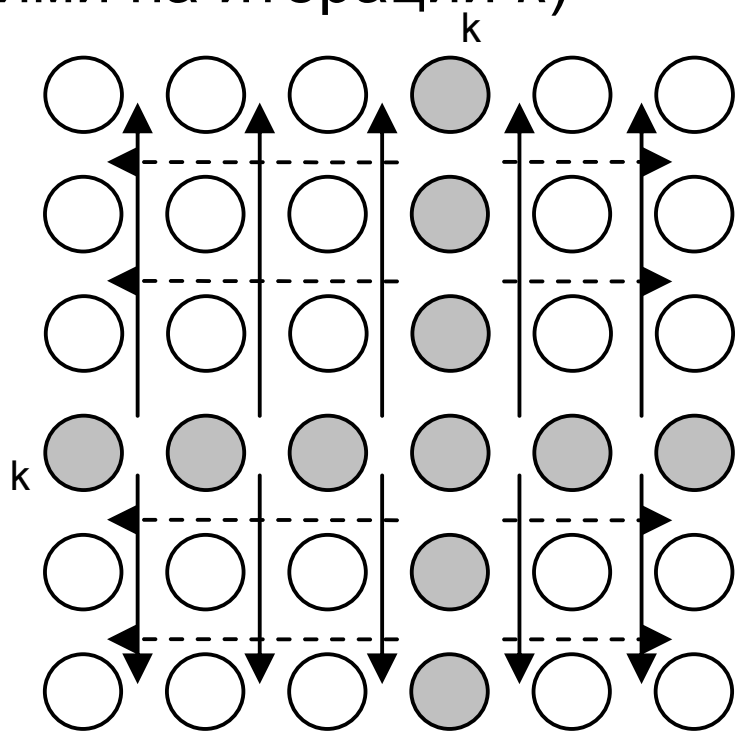
    - Аналогично для j=k

Задача поиска всех кратчайших путей

- **Выделение информационных зависимостей**
  - Выполнение вычислений в подзадачах становится возможным только тогда, когда каждая подзадача (*i,j*) содержит необходимые для расчетов элементы *Aij*, *Aik*, *Akj* матрицы *A*
  - Каждый элемент *Akj* строки *k* матрицы *A* должен быть передан всем подзадачам (*k,j*), $1 \le j \le n$, а каждый элемент *Aik* столбца *k* матрицы *A* должен быть передан всем подзадачам (*i,k*), $1 \le i \le n$

Задача поиска всех кратчайших путей

- **Выделение информационных зависимостей**
  - Информационная зависимость базовых подзадач (стрелками показаны направления обмена значениями на итерации *k*)



Задача поиска всех кратчайших путей

- **Масштабирование и распределение подзадач по процессорам**
  - Возможный способ укрупнения вычислений состоит в использовании *ленточной схемы* разбиения матрицы *A*
    - обновление элементов одной или нескольких строк (*горизонтальное* разбиение)
    - обновление элементов одной или нескольких строк (*вертикальное* разбиение)
  - Далее будем рассматривать только разбиение матрицы *A* на горизонтальные полосы

Задача поиска всех кратчайших путей

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{n^3}{(n^3/p)} = p \qquad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1$$

Разработанный способ параллельных вычислений позволяет достичь идеальных показателей ускорения и эффективности

Задача поиска всех кратчайших путей

❑ Анализ эффективности (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = n^2 \cdot \lceil n/p \rceil \cdot \tau$$

- Оценка трудоемкости выполняемых операций передачи данных может быть определена как

$$T_p(comm) = n\lceil \log_2 p \rceil(\alpha + wn/\beta)$$

(предполагается, что все операции передачи данных между процессорами в ходе одной итерации алгоритма могут быть выполнены параллельно)

Общее время выполнения параллельного алгоритма составляет

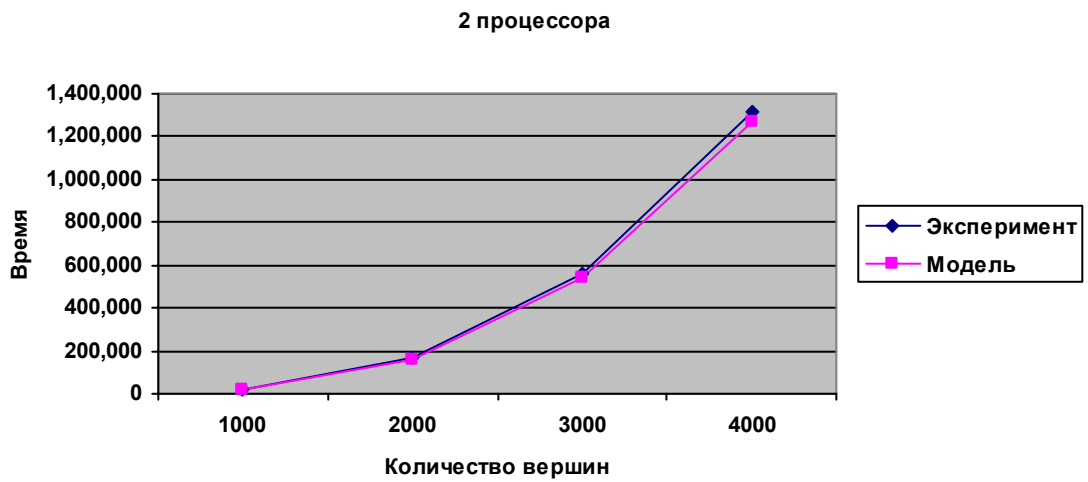
$$T_p = n^2 \cdot \lceil n/p \rceil \cdot \tau + n \cdot \lceil \log_2(p) \rceil(\alpha + w \cdot n/\beta)$$

Задача поиска всех кратчайших путей

Результаты вычислительных экспериментов

– Сравнение теоретических оценок Tp и экспериментальных данных Tp*

Количество вершин	Последовательный алгоритм		Параллельный алгоритм					
			2 процессора		4 процессора		8 процессоров	
	T_1	T_1^*	T_2	T_2^*	T_4	T_4^*	T_8	T_8^*
1000	41,600	39,686	21,800	20,996	11,900	10,530	6,910	6,590
2000	317,000	306,228	159,000	166,219	80,500	83,468	41,400	48,796
3000	1070,000	1027,611	537,000	557,887	270,000	279,830	137,000	160,013
4000	2540,000	2445,473	1270,000	1320,205	639,000	661,641	323,000	336,713

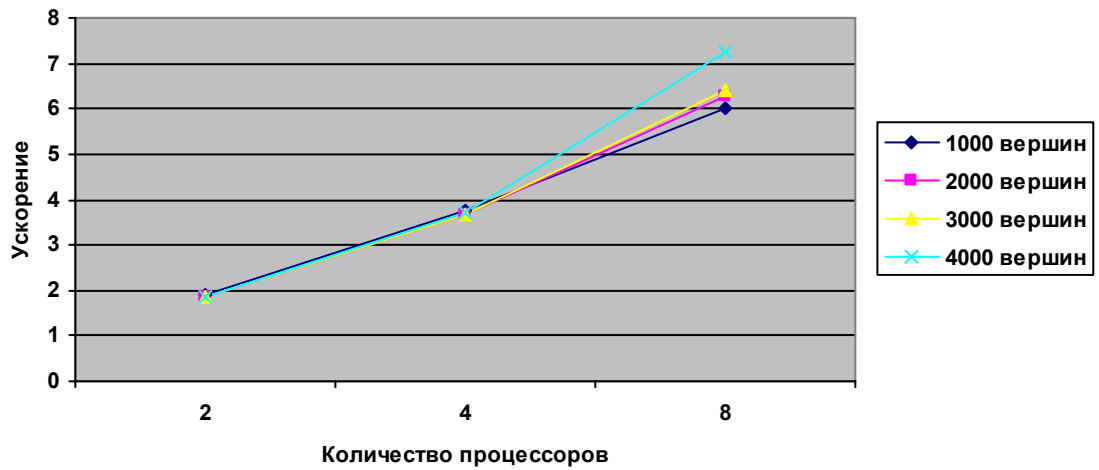


Задача поиска всех кратчайших путей

Результаты вычислительных экспериментов

– Ускорение вычислений

Количество вершин	Последовательный алгоритм	Параллельный алгоритм					
		2 процессора		4 процессора		8 процессора	
		Время	Ускорение	Время	Ускорение	Время	Ускорение
1000	39,686	20,996	1,890	10,530	3,769	6,590	6,022
2000	306,228	166,219	1,842	83,468	3,669	48,796	6,276
3000	1027,611	557,887	1,842	279,830	3,672	160,013	6,422
4000	2445,473	1320,205	1,852	661,641	3,696	336,713	7,263

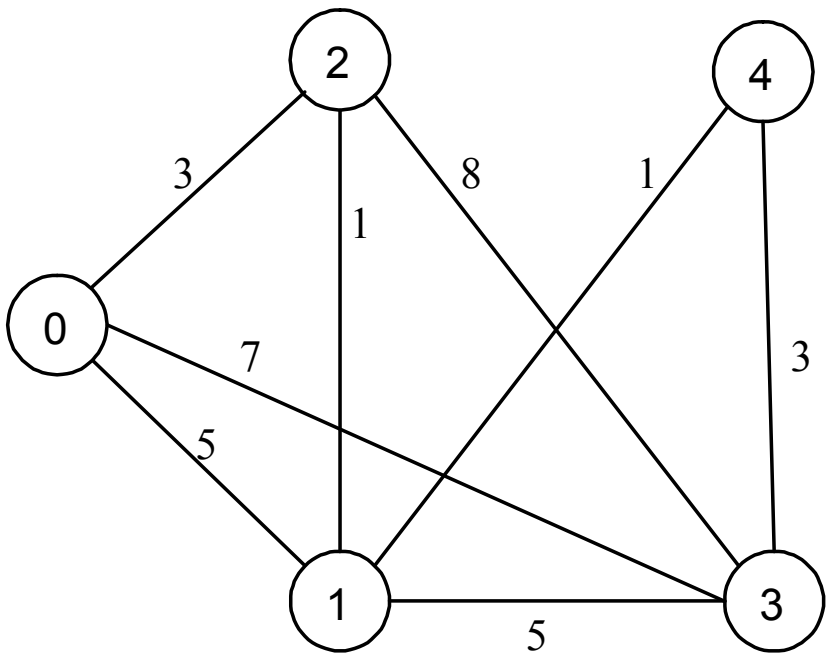


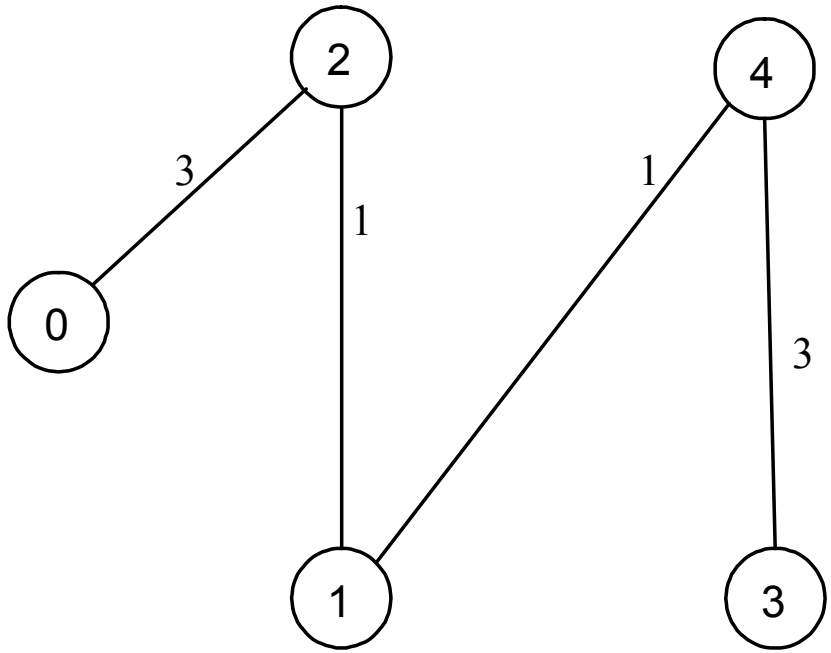
Задача нахождения минимального охватывающего дерева

- **Постановка задачи**:
  - *Охватывающим деревом* (или *остовом*) неориентированного графа *G* называется подграф *T* графа *G*, который является деревом и содержит все вершины из *G*
  - Определив вес подграфа для взвешенного графа как сумму весов входящих в подграф дуг, под *минимально охватывающим деревом (МОД) T* будем понимать охватывающее дерево минимального веса
  - Таким образом, для данного графа G требуется найти минимальное охватывающее дерево T

Задача нахождения минимального охватывающего дерева

- Пример нахождения МОД





Задача нахождения минимального охватывающего дерева

Последовательный алгоритм Прима

– Пусть V_T есть множество вершин, уже включенных алгоритмом в МОД, а величины d_i, $1 \le i \le n$ характеризуют дуги минимальной длины от вершин, еще не включенных в дерево, до множества V_T, т.е.

$$\forall i \notin V_T \Rightarrow d_i = \min\{w(i,u) : u \in V_T, (i,u) \in R\}$$

(если для какой-либо вершины i не существует ни одной дуги в V_T, значение d_i устанавливается в ∞)

– В начале работы алгоритма выбирается корневая вершина МОД s и полагается $V_T=\{s\}$, $d_s=0$.

Задача нахождения минимального охватывающего дерева

Последовательный алгоритм Прима

- Действия, выполняемые на каждой итерации алгоритма Прима, состоят в следующем:
  - определяются значения величин d_i для всех вершин, еще не включенных в состав МОД
  - выбирается вершина *t* графа *G*, имеющая дугу минимального веса до множества V_T

$$t : d_t = \min(d_i),\ i \notin V_T$$

  - вершина *t* включается в V_T
- Трудоемкость алгоритма Прима характеризуется квадратичной зависимостью от числа вершин графа

$$O(n^2)$$

Задача нахождения минимального охватывающего дерева

- **Разделение вычислений на независимые части**
  - Действия, выполняемые на каждой итерации алгоритма, являются независимыми и могут реализовываться одновременно
  - Каждый процессор при равномерной загрузке должен содержать:
    - Набор вершин

$$V_j = \{v_{i_j+1}, v_{i_j+2}, \ldots, v_{i_j+k}\}, \; i_j = k \cdot (j-1), k = \lceil n/p \rceil$$

    - Соответствующий этому набору блок из k величин

$$\Delta_j = \{d_{i_j+1}, d_{i_j+2}, \ldots, d_{i_j+k}\}$$

    - Вертикальную полосу матрицы смежности графа G из k соседних столбцов

$$A_j = \{\alpha_{i_j+1}, \alpha_{i_j+2}, \ldots, \alpha_{i_j+k}\}$$

    (α_s есть s-ый столбец матрицы A)

    - общую часть набора V_j и формируемого в процессе вычислений множества вершин V_T

Задача нахождения минимального охватывающего дерева

- **Выделение информационных зависимостей**
  - Общая схема параллельного выполнения алгоритма Прима будет состоять в следующем:
    - Определяется вершина *t* графа *G*, имеющая дугу минимального веса до множества V_T; для выбора такой вершины необходимо осуществить поиск минимума в наборах величин d_i, имеющихся на каждом из процессоров, и выполнить сборку полученных значений на одном из процессоров
    - Номер выбранной вершины для включения в охватывающее дерево передается всем процессорам
    - Обновляются наборы величин d_i с учетом добавления новой вершины
  - Таким образом, в ходе параллельных вычислений между процессорами выполняются два типа информационных взаимодействий - сбор данных от всех процессоров на одном из процессоров и передача сообщений от одного процессора всем процессорам вычислительной системы

Задача нахождения минимального охватывающего дерева

- **Масштабирование и распределение подзадач по процессорам**
  - Распределение подзадач между процессорами должно учитывать характер выполняемых в алгоритме Прима коммуникационных операций. Для эффективной реализации требуемых информационных взаимодействий между базовыми подзадачами топология сети передачи данных должна иметь структуру гиперкуба или полного графа

Задача нахождения минимального охватывающего дерева

- **Анализ эффективности**
  - Общая оценка показателей ускорения и эффективности

$$S_p = \frac{n^3}{(n^3/p)} = p \qquad E_p = \frac{n^3}{p \cdot (n^3/p)} = 1$$

Задача нахождения минимального охватывающего дерева

❑ **Анализ эффективности** (уточненные оценки)

- Время выполнения параллельного алгоритма, связанное непосредственно с вычислениями, составляет

$$T_p(calc) = n\lceil n/p \rceil \cdot \tau$$

- Оценка трудоемкости выполняемых операций передачи данных может быть определена как

$$T_p^1(comm) = n(\alpha \log_2 p + 3w(p-1)/\beta)$$
$$T_p^2(comm) = n \log_2 p(\alpha + w/\beta)$$

Общее время выполнения параллельного алгоритма составляет

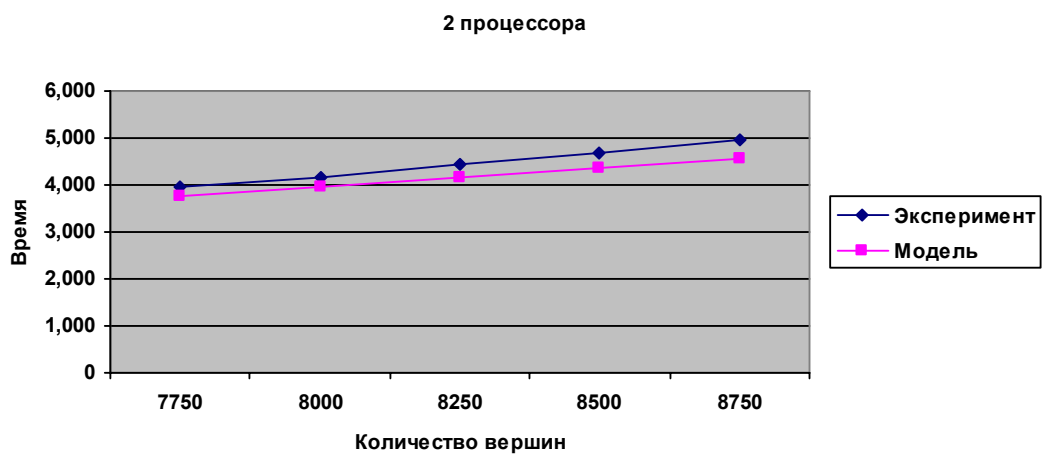
$$T_p = n\lceil n/p \rceil \cdot \tau + n(\alpha \cdot \log_2 p + 3w(p-1)/\beta + \log_2 p(\alpha + w/\beta))$$

Задача нахождения минимального охватывающего дерева

Результаты вычислительных экспериментов

– Сравнение теоретических оценок Tp и экспериментальных данных Tp*

Количество вершин	Последовательный алгоритм		Параллельный алгоритм					
			2 процессора		4 процессора		6 процессоров	
	T_1	T_1^*	T_2	T_2^*	T_4	T_4^*	T_6	T_6^*
7750	4,234	4,137	3,778	3,971	4,383	3,462	5,005	3,837
8000	4,512	4,498	3,971	4,148	4,560	3,621	5,190	3,976
8250	4,798	4,800	4,168	4,442	4,739	3,763	5,376	4,120
8500	5,094	5,206	4,369	4,680	4,920	3,992	5,564	4,241
8750	5,398	5,689	4,574	4,950	5,103	4,623	5,754	4,424

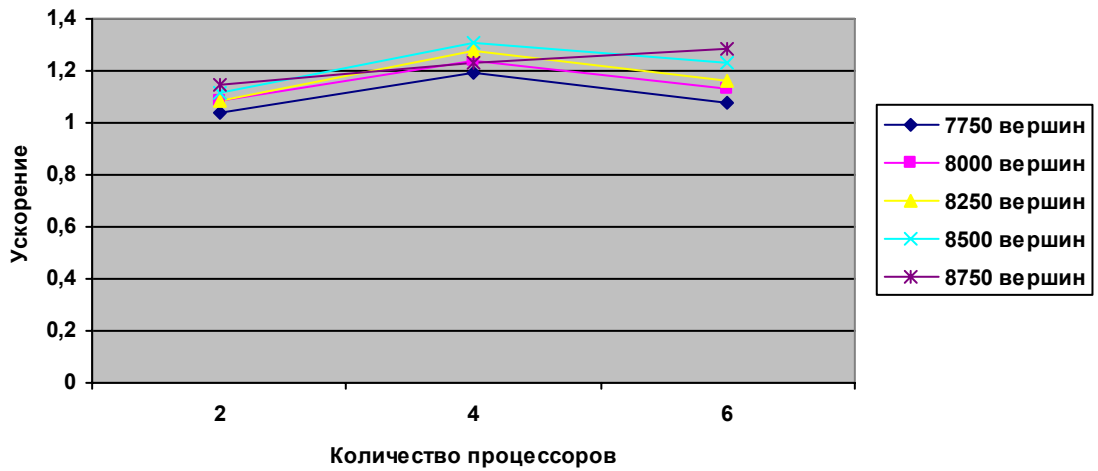


Задача нахождения минимального охватывающего дерева

Результаты вычислительных экспериментов

- Ускорение вычислений

Количество вершин	Последовательный алгоритм	Параллельный алгоритм					
		2 процессора		4 процессора		6 процессора	
	Время	Время	Ускорение	Время	Ускорение	Время	Ускорение
7750	4,137	3,971	1,042	3,462	1,195	3,837	1,078
8000	4,498	4,148	1,084	3,621	1,242	3,976	1,131
8250	4,800	4,442	1,081	3,763	1,276	4,120	1,165
8500	5,206	4,680	1,112	3,992	1,304	4,241	1,228
8750	5,689	4,950	1,149	4,623	1,231	4,424	1,286

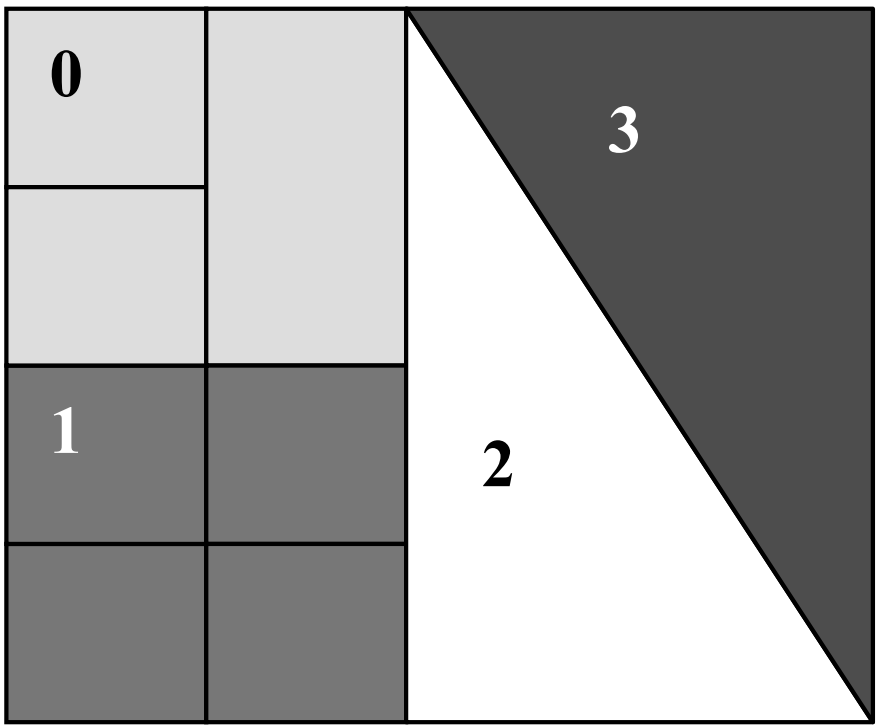


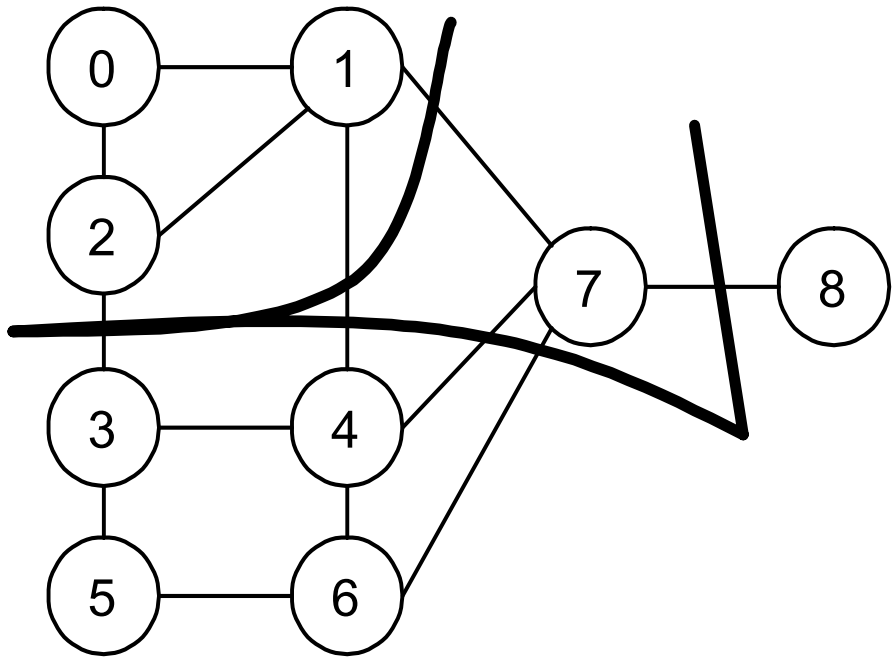
Задача оптимального разделения графов

- **Введение**
  - Проблема оптимального разделения графов относится к числу часто возникающих задач при проведении различных научных исследований, использующих параллельные вычисления. В качестве примера можно привести задачи обработки данных, в которых области расчетов представляются в виде двухмерной или трехмерной сети
  - Задачи разделения *вычислительной сети* между процессорами могут быть сведены к проблеме оптимального разделения графа
  - Для представления сети в виде графа каждому элементу сети можно поставить в соответствие вершину графа, а дуги графа использовать для отражения свойства близости элементов сети (например, определять дуги между вершинами графа тогда и только тогда, когда соответствующие элементы исходной сети являются соседними)

Задача оптимального разделения графов

- **Пример разделения нерегулярной сети и соответствующий сети граф:**





Задача оптимального разделения графов

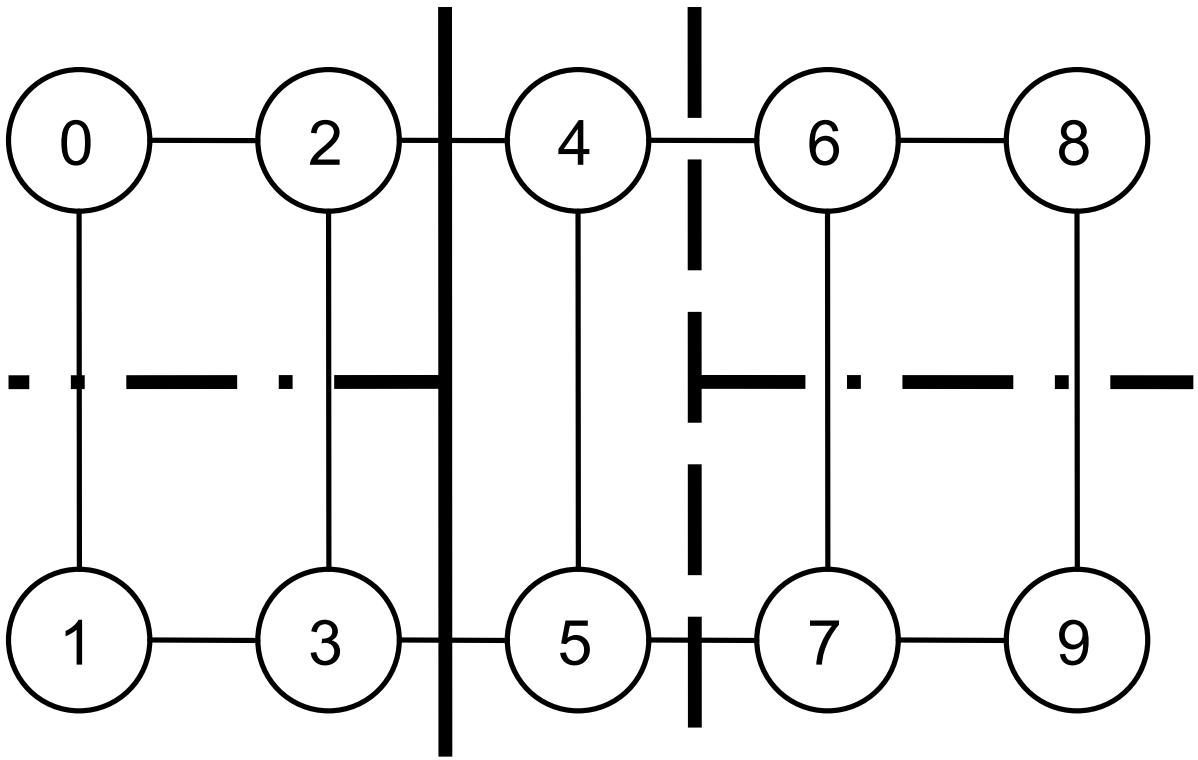
- **Постановка задачи**
  - Пусть дан взвешенный неориентированный граф $G=(V,E)$, каждой вершине v и каждому ребру e которого приписан вес
  - Задача оптимального разделения графа состоит в разбиении его вершин на непересекающиеся подмножества с максимально близкими суммарными весами вершин и минимальным суммарным весом ребер, проходящих между полученными подмножествами вершин
  - Далее будем полагать веса вершин и ребер графа равными единице

Задача оптимального разделения графов

- **Метод рекурсивного деления пополам**
  - Для решения задачи разбиения графа можно рекурсивно применить *метод бинарного деления*, при котором на первой итерации граф разделяется на две равные части, далее на втором шаге каждая из полученных частей также разбивается на две части и т.д.
  - В данном подходе для разбиения графа на *k* частей необходимо $\log_2(k)$ уровней рекурсии и выполнение *k-1* деления пополам. В случае, когда требуемое количество разбиений *k* не является степенью двойки, каждое деление пополам необходимо осуществлять в соответствующем соотношении

Задача оптимального разделения графов

- **Пример разбиения графа на 5 частей методом рекурсивного деления пополам**

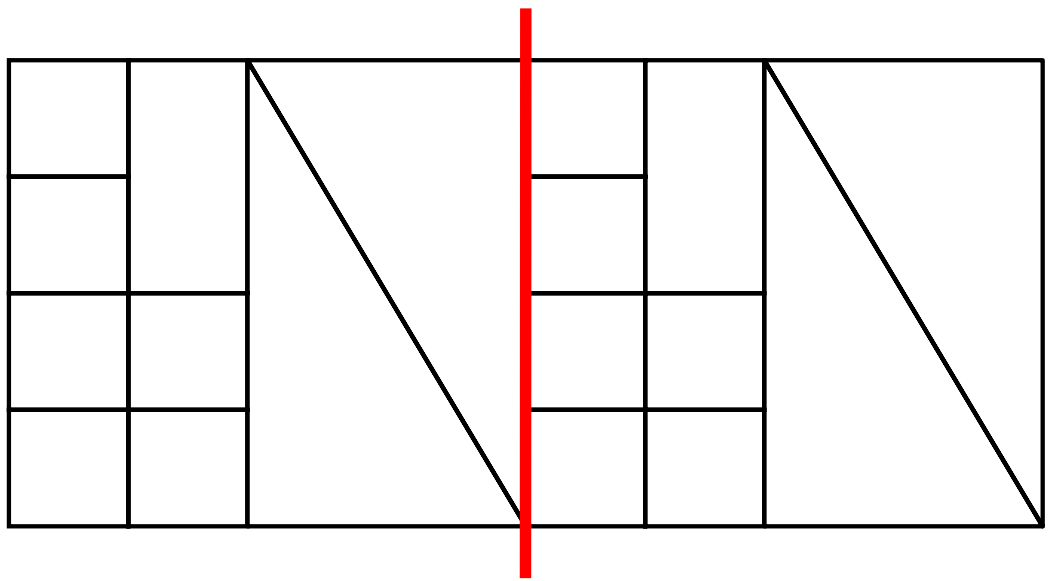


Задача оптимального разделения графов

- **Геометрические методы**
  - Геометрические методы выполняют разбиение сетей, основываясь исключительно на координатной информации об узлах сети
  - Так как геометрические методы не принимают во внимание информацию о связности элементов сети, то они не могут явно привести к минимизации суммарного веса граничных ребер. Для минимизации межпроцессорных коммуникаций геометрические методы оптимизируют некоторые вспомогательные показатели (например, длину границы между разделенными участками сети)
  - Геометрические методы работают исключительно быстро
  - Рассматриваемые геометрические методы:
    - Покоординатное разбиение
    - Рекурсивный инерционный метод деления пополам
    - Деление сети с использованием кривых Пеано

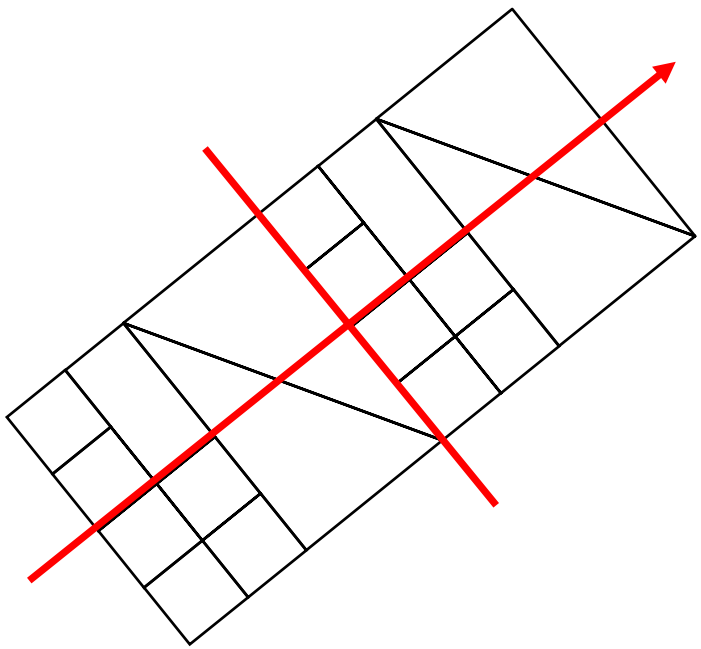
Задача оптимального разделения графов

- **Покоординатное разбиение**
  - Общая схема выполнения метода:
    - Вычисляются центры масс элементов сети
    - Полученные точки проектируются на ось, соответствующую наибольшей стороне разделяемой сети



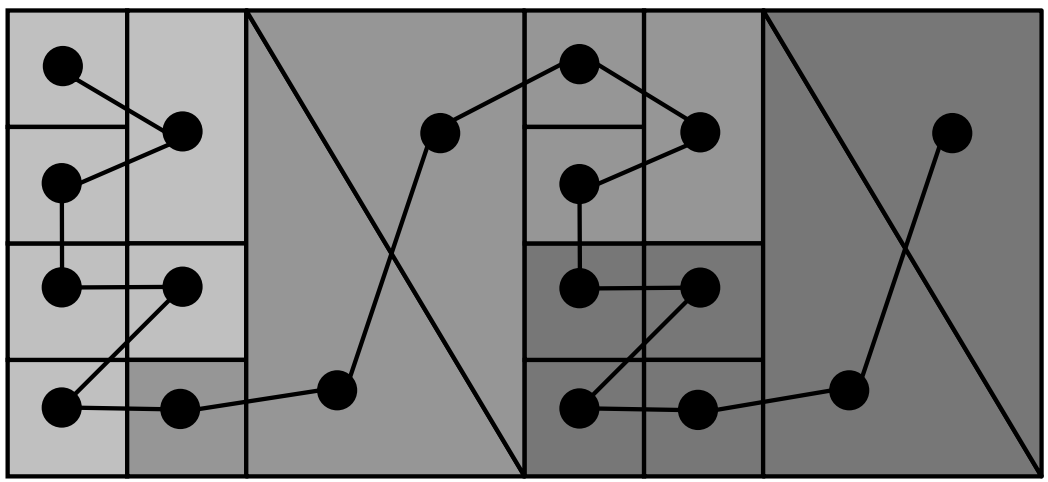
Задача оптимального разделения графов

- **Рекурсивный инерционный метод деления пополам**
  - Рекурсивном инерционном методе деления пополам строит главную инерционную ось, считая элементы сети точечными массами
  - Линия бисекции, ортогональная полученной оси, как правило, дает границу наименьшей длины



Задача оптимального разделения графов

- **Деление сети с использованием кривых Пеано**
  - Кривые Пеано – это кривые, полностью заполняющие фигуры больших размерностей (например, квадрат или куб)
  - После получения списка элементов сети, упорядоченного в соответствии с расположением на кривой, достаточно разделить список на необходимое число частей в соответствии с установленным порядком



Задача оптимального разделения графов

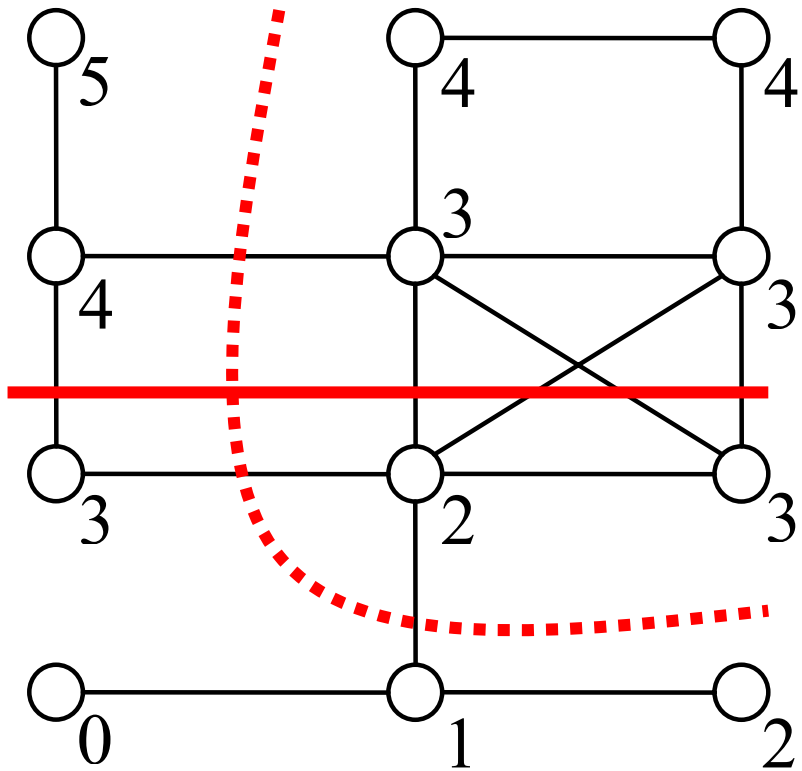
- **Комбинаторные методы**
  - В отличие от геометрических методов, комбинаторные алгоритмы обычно оперируют не с сетью, а с графом, построенным для этой сети
  - Комбинаторные методы не принимают во внимание информацию о близости расположения элементов сети друг относительно друга, руководствуясь только смежностью вершин графа
  - Комбинаторные методы обеспечивают более сбалансированное разбиение и меньшее информационное взаимодействие полученных подсетей по сравнению с геометрическими методами
  - Время работы комбинаторных методов, как правило, существенно превосходит времена работы геометрических
  - Рассматриваемые геометрические методы:
    - Деление с учетом связности
    - Алгоритм Кернигана-Лина

Задача оптимального разделения графов

- **Деление с учетом связности**
  - На каждой итерации алгоритма происходит разделение графа на 2 части. Разделение графа на требуемое число частей достигается путем рекурсивного применения алгоритма
  - Общая схема алгоритма:
    1. Iteration = 0
    2. Присвоение номера Iteration произвольной вершине графа
    3. Присвоение ненумерованным соседям вершин с номером Iteration номера Iteration + 1
    4. Iteration = Iteration + 1
    5. Если еще есть неперенумерованные соседи, то переход на шаг 3
    6. Разделение графа на 2 части в порядке нумерации

Задача оптимального разделения графов

- **Пример работы алгоритма деления с учетом связности**

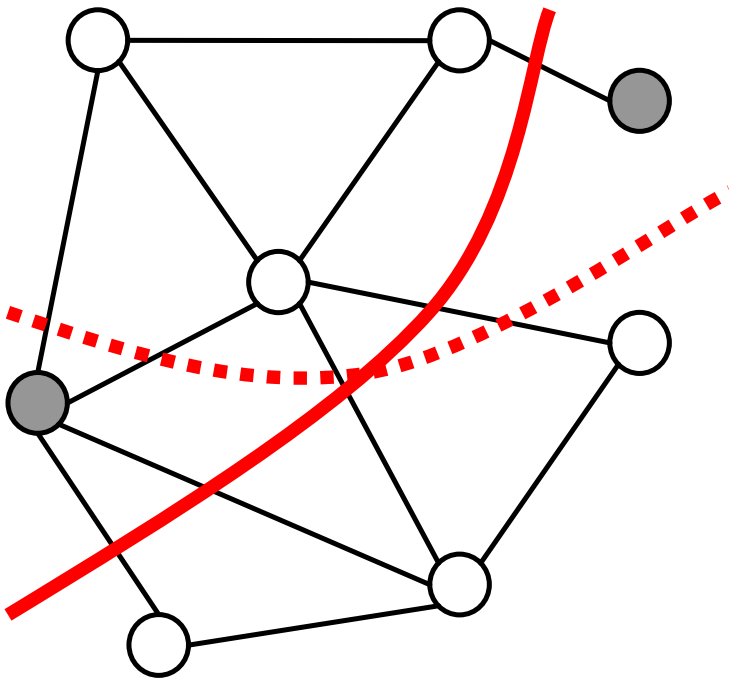


Задача оптимального разделения графов

- **Алгоритм Кернигана-Лина**
  - В *алгоритме* Кернигана-Лина при начале работы метода предполагается, что некоторое начальное разбиение графа уже существует, затем же имеющееся приближение улучшается в течение некоторого количества итераций
  - Общая схема алгоритма:
    1. Из вершин, которые еще не были переставлены на данной итерации, выбирается пара (по одной из каждого подграфа), которая обеспечивает наилучшее деление графа (даже если полученное деление хуже, чем было до перестановки пары вершин. Происходит перестановка
    2. Если есть вершины, которые еще не были переставлены на данной итерации, то переход на шаг 1 (перестановка очередной пары вершин), иначе переход на шаг 3
    3. Среди всех разбиений графа, рассмотренных на данной итерации, выбирается (и фиксируется) наилучший вариант

Задача оптимального разделения графов

- **Пример работы алгоритма Кернигана-Лина**
  - В приведенном примере переставляются 2 вершины (выделены серым)



Задача оптимального разделения графов

❑ Сравнительная таблица некоторых алгоритмов разделения графов

		Необходимость координатной информации	Точность	Время выполнения	Возможности для распарал-лелив ания
Покоординатное разбиение		да	●	●	●●●
Рекурсивный инерционный метод деления пополам		да	●●	●	●●●
Деление с учетом связности		нет	●●	●●	●●
Алгоритм Кернигана-Лина	1 итерация	нет	●●	●●	●
	10 итераций	нет	●●●◐	●●●	●●
	50 итераций	нет	●●●●		●●

Заключение

- В разделе был рассмотрены ряд алгоритмов для решения типовых задач обработки графов:
  - Алгоритм Флойда
  - Алгоритм Прима
- Был приведен обзор методов разделения графа:
  - Геометрические методы:
    - Покоординатное разбиение
    - Рекурсивный инерционный метод деления пополам
    - Деление сети с использованием кривых Пеано
  - Комбинаторные методы:
    - Деление с учетом связности
    - Алгоритм Кернигана-Лина

Вопросы для обсуждения

- Приведите определение графа. Какие основные способы используются для задания графов?
- В чем состоит задача поиска всех кратчайших путей?
- Приведите общую схему алгоритма Флойда. Какова трудоемкость алгоритма?
- В чем состоит способ распараллеливания алгоритма Флойда?
- В чем заключается задача нахождения минимального охватывающего дерева? Приведите пример использования задачи на практике.
- Приведите общую схему алгоритма Прима. Какова трудоемкость алгоритма?
- В чем состоит способ распараллеливания алгоритма Прима?
- В чем отличие геометрических и комбинаторных методов разделения графа? Какие методы являются более предпочтительными? Почему?
- Приведите описание метода покоординатного разбиения и алгоритма разделения с учетом связности. Какой из этих методов является более простым для практической реализации?

Темы заданий для самостоятельной работы

- Используя приведенный программный код, выполните реализацию параллельного алгоритма Флойда. Проведите вычислительные эксперименты. Постройте теоретические оценки с учетом параметров используемой вычислительной системы. Сравните полученные оценки с экспериментальными данными.
- Выполните реализацию параллельного алгоритма Прима. Проведите вычислительные эксперименты. Постройте теоретические оценки с учетом параметров используемой вычислительной системы. Сравните полученные оценки с экспериментальными данными.
- Разработайте программную реализацию алгоритма Кернигана – Лина. Дайте оценку возможности распараллеливания этого алгоритма.

Литература

- **Cormen,** T.H., Leiserson, C. E. , Rivest, R. L. , Stein C. (2001). Introduction to Algorithms, 2nd Edition. - The MIT Press. (русский перевод Кормен Т., Лейзерсон Ч., Ривест Р. (1999). Алгоритмы: построение и анализ. – М.: МЦНТО.)
- **Schloegel,** K., Karypis, G., Kumar, V. (2000). Graph Partitioning for High Performance Scientific Simulations.

Следующая тема

- !!! A. V. S.2

A. V. S.2 Добавить!
Andrew Senin; 11.08.2005

Авторский коллектив

Гергель В.П., профессор, д.т.н., руководитель

Гришагин В.А., доцент, к.ф.м.н.

Абросимова О.Н., ассистент (раздел 10)

Лабутин Д.Ю., ассистент (система ПараЛаб)

Курылев А.Л., ассистент (лабораторные работы 4, 5)

Сысоев А.В., ассистент (раздел 1)

Гергель А.В., аспирант (раздел 12, лабораторная работа 6)

Лабутина А.А., аспирант (разделы 7,8,9, лабораторные работы 1, 2, 3, система ПараЛаб)

Сенин А.В., аспирант (раздел 11, лабораторные работы по Microsoft Compute Cluster)

Ливерко С.В. (система ПараЛаб)

О проекте

Целью проекта является создание образовательного комплекса "Многопроцессорные вычислительные системы и параллельное программирование", обеспечивающий рассмотрение вопросов параллельных вычислений, предусматриваемых рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM. Данный образовательный комплекс может быть использован для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы параллельного программирования" и лабораторный практикум "Методы и технологии разработки параллельных программ"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения для решения сложных вычислительно-трудоемких задач на высокопроизводительных вычислительных системах.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (http://www.software.unn.ac.ru). Выполнение проекта осуществлялось при поддержке компании Microsoft.