Параллельная обработка больших графов

Занятие 10

А.С. Семенов

dislab.org

НИЦ ВТ

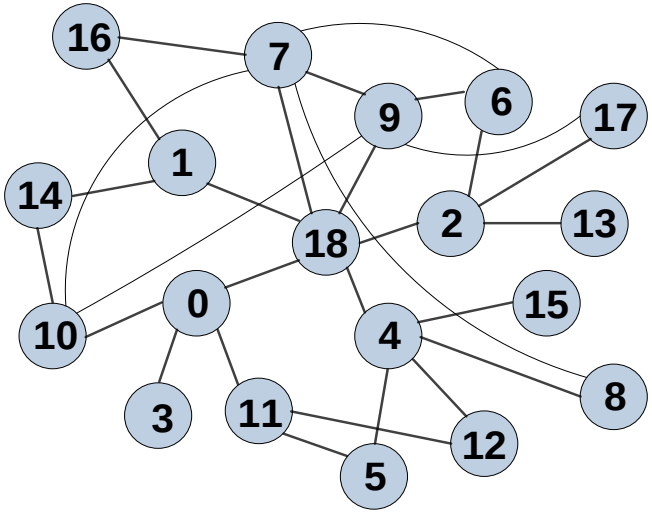
DISLab

Проблемы и подходы к решению графовых задач на распределенной памяти

Проблемы анализа больших графов

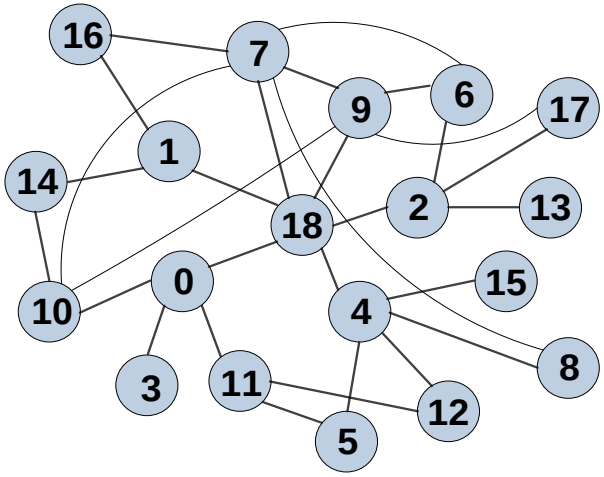
- **Data-driven computations.** Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.
- **Unstructured problems.** Работа с нерегулярными, неструктурированными данными, трудность распараллеливания.
- **Poor locality.** Низкая пространственно-временная локализация обращений к памяти.
- **High data access to computation ratio.** Преобладание команд доступа к памяти над командами выполнения арифметических операций.

Представление графа

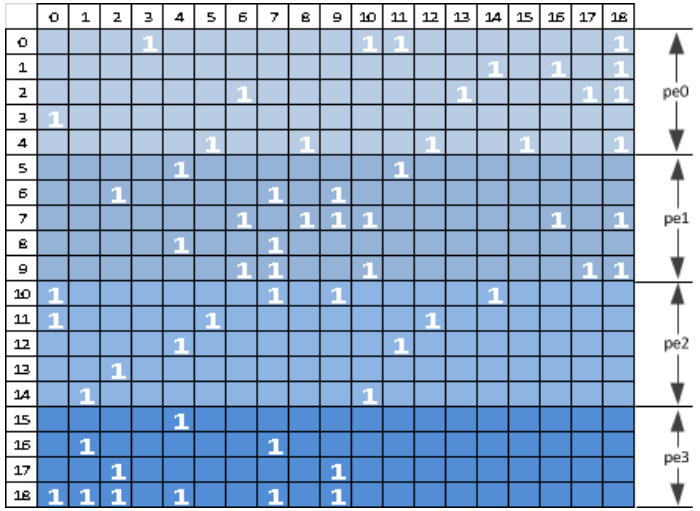


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1							1		1					1				
11	1					1							1						
12					1							1							
13			1																
14		1									1								
15					1														
16		1						1											
17			1							1									
18	1	1	1		1			1		1									

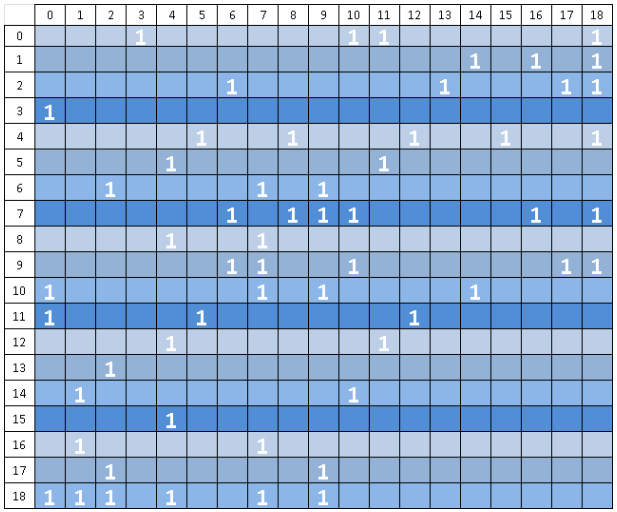
Распределение данных



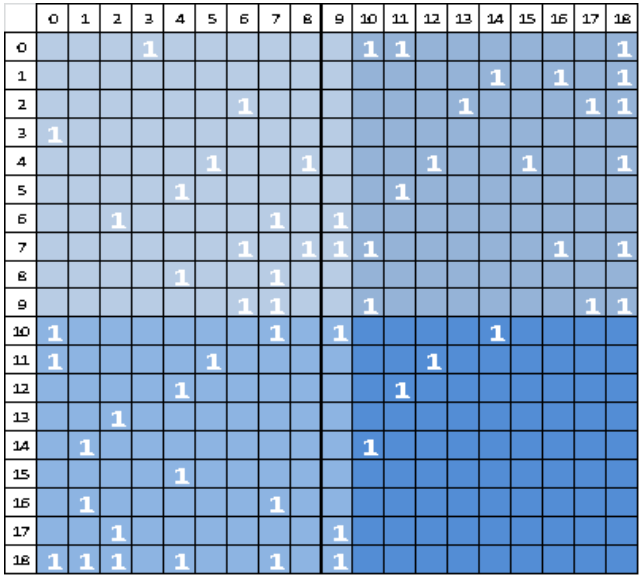
1D, блоками



1D, с чередованием

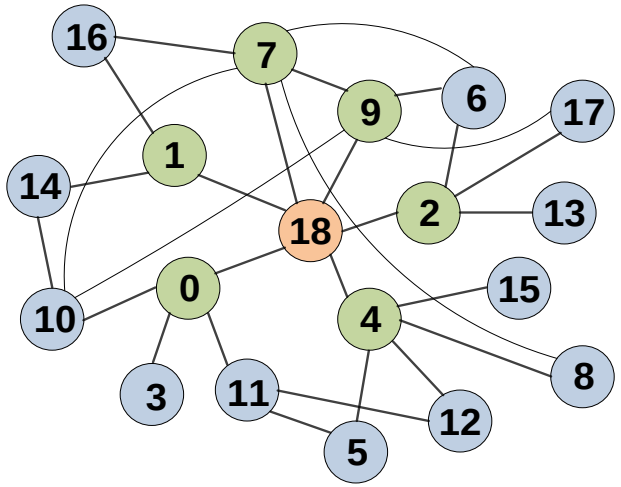


2D



Поиск вширь в графе, распределенная версия

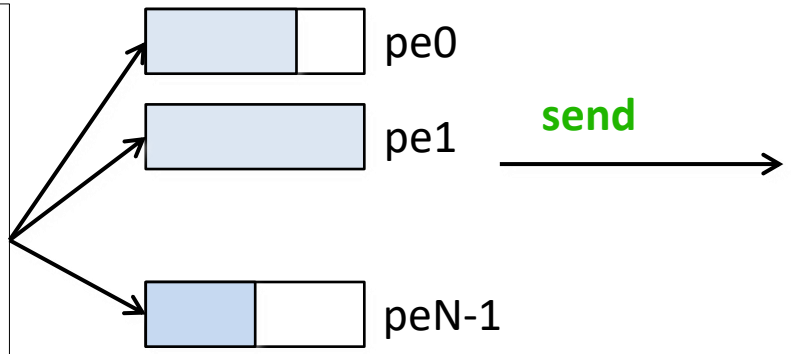
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

Поиск вширь в графе, агрегация сообщений

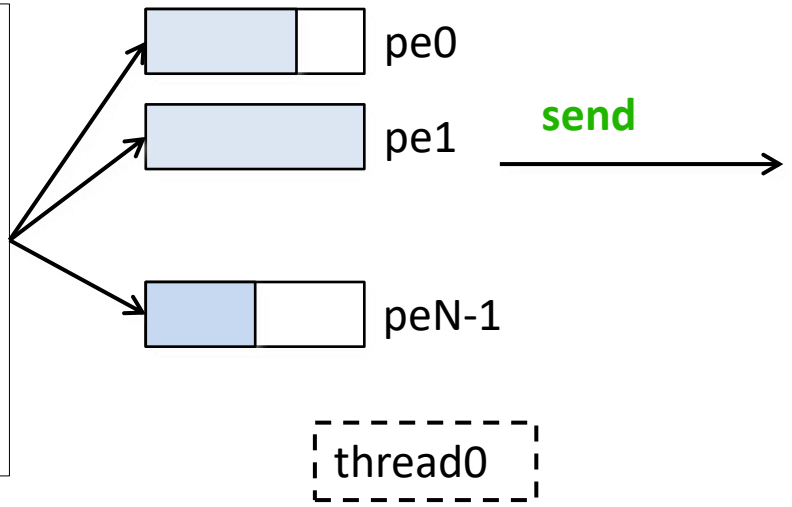
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

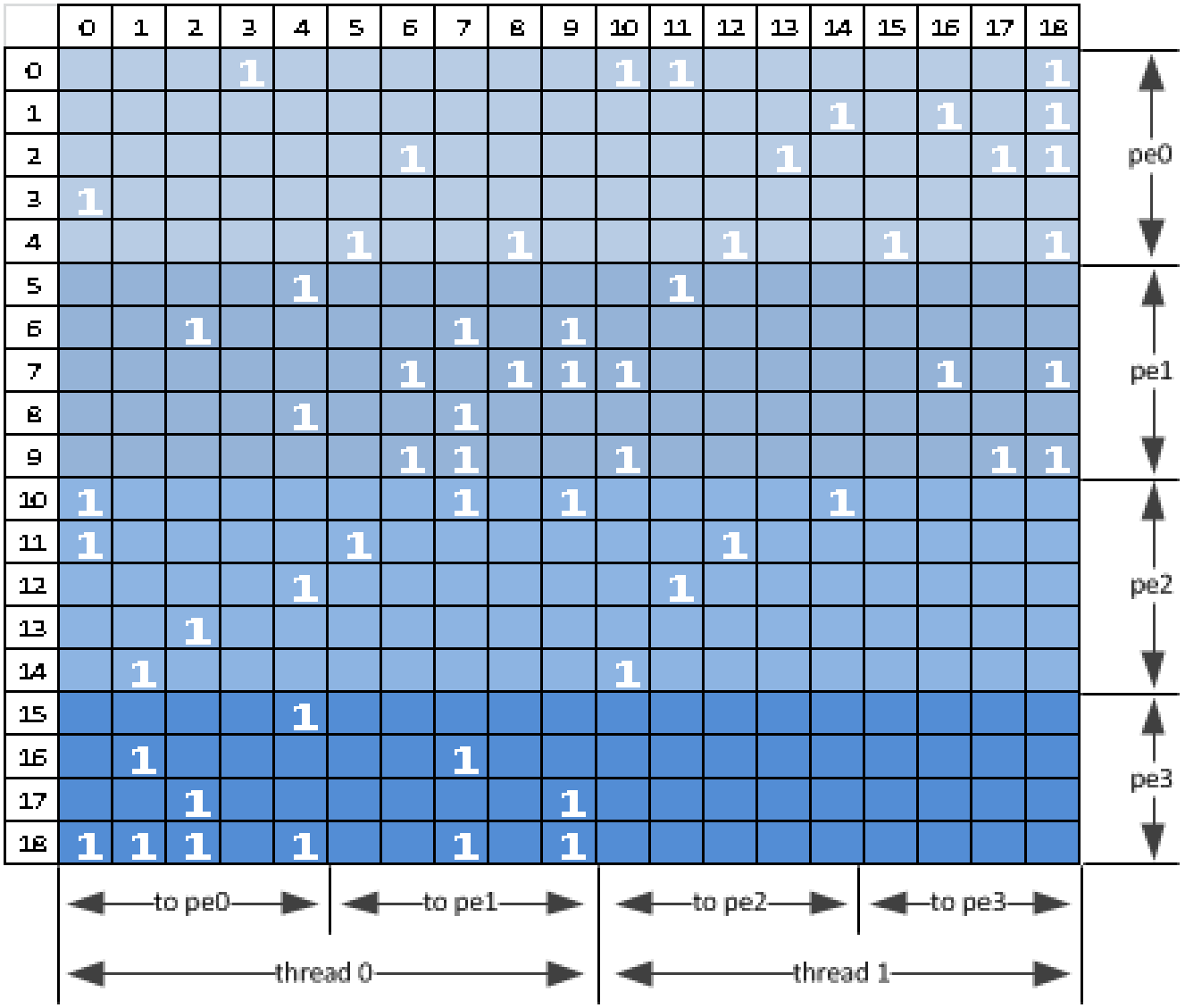
Поиск вширь в графе, параллельная отправка и прием

```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```

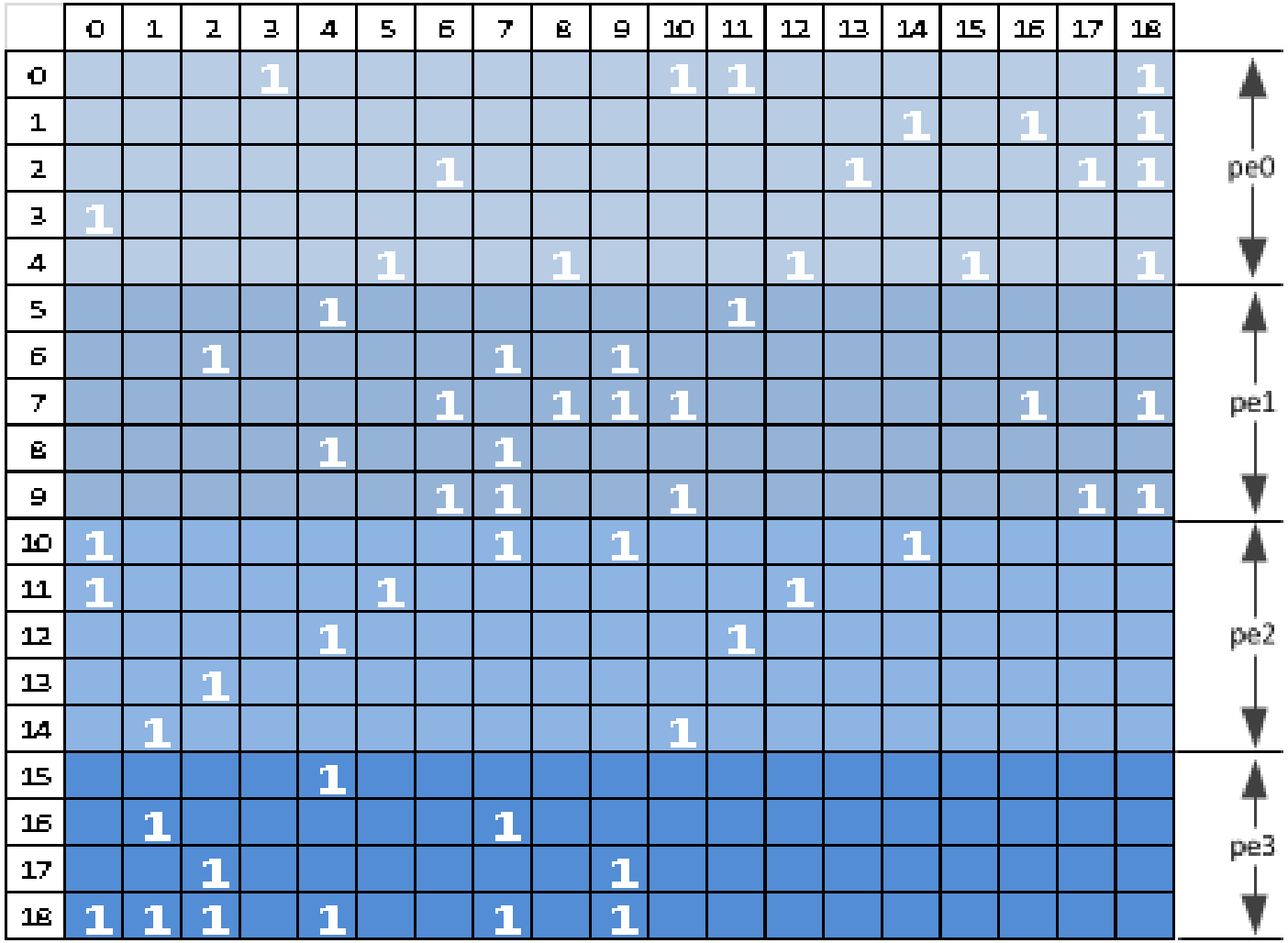



```
function Receive(vertex, w)
    if w ∉ Visited then
        Q_next = Q_next ∪ w
        Visited = Visited ∪ w
        Parents(w) = vertex
    end if
end function
```

Организация параллелизма потоков



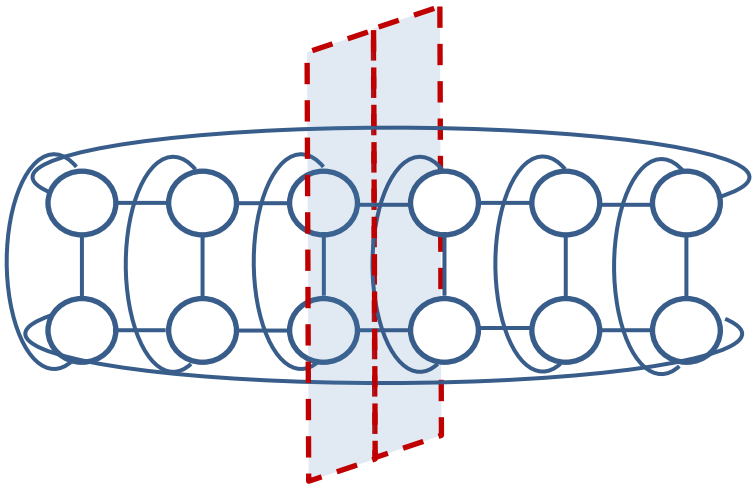
Хаотично расположенные вершины и ребра графа



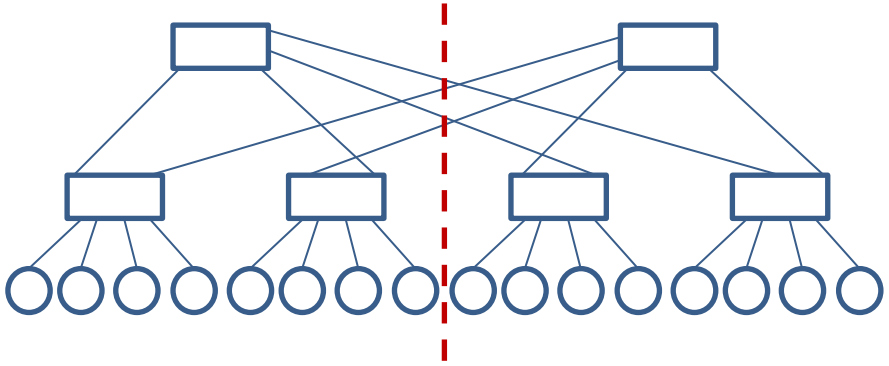
Шаблон обменов all-to-all

Коммуникационная сеть. Бисекционная пропускная способность

- **Бисекционная плоскость – минимальный разрез, который разделяет сеть на две равные связные части**
- Бисекционная пропускная способность – пропускная способность каналов связи через бисекционную плоскость
- **В случае равномерных случайных посылок (all-to-all) каждый узел посылает сообщение через бисекционную плоскость с вероятностью ½**
- **Посылают все узлы – для линейной масштабируемости требуется N/2 линков в бисекционной плоскости**



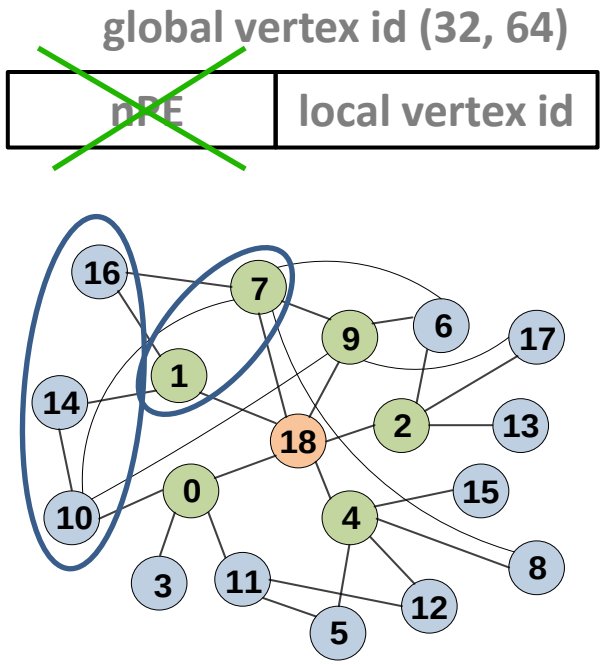
Бисекция тора = 2N/Nmax



Бисекция жирного дерева (half bisection) = N/4

Уменьшение количества пересылаемых данных

- **Использование простаивающего процессора**
- **Сокращение пересылок**
  - **Отказ от лишней пересылаемой информации**
  - **Удаление дублирующей информации**
- **Сжатие данных**
  - **Использование знаний о структуре графа**



Графы реального мира. Степенной закон

- WWW, Социальные сети, Биоинформатика
- Графы small-world

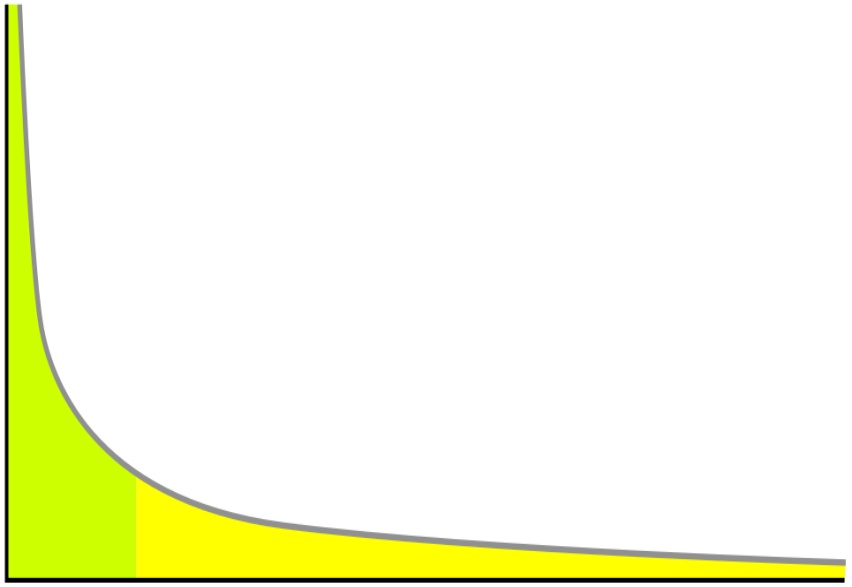
L ~ log N,

- scale-free – графы,

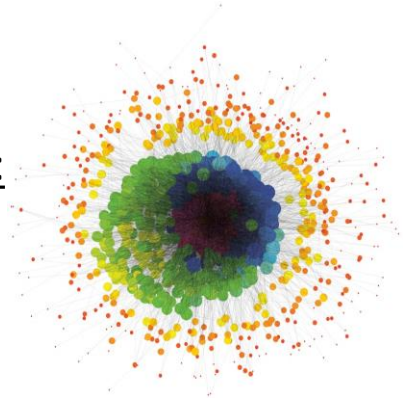
доля $P(k) \sim k^{-tau}$, $2 < tau < 3$

k – связность вершины

L ~ log log N



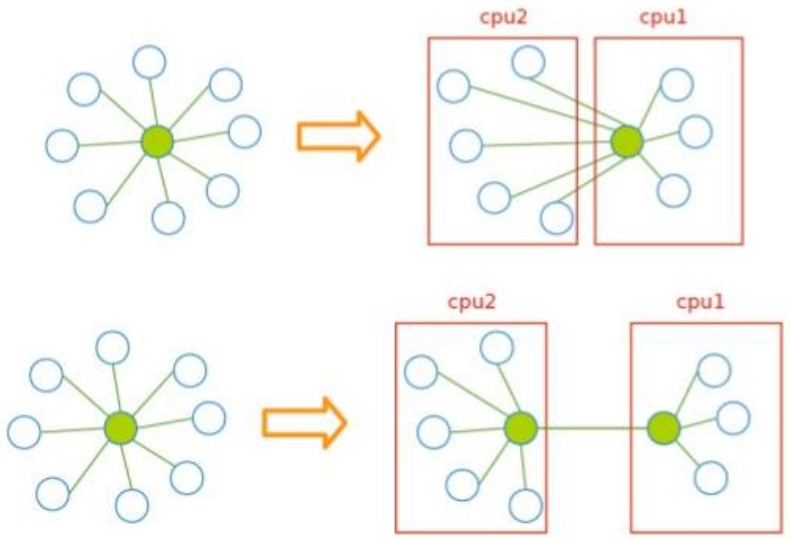
Граф Кронекера:



Балансировка нагрузки

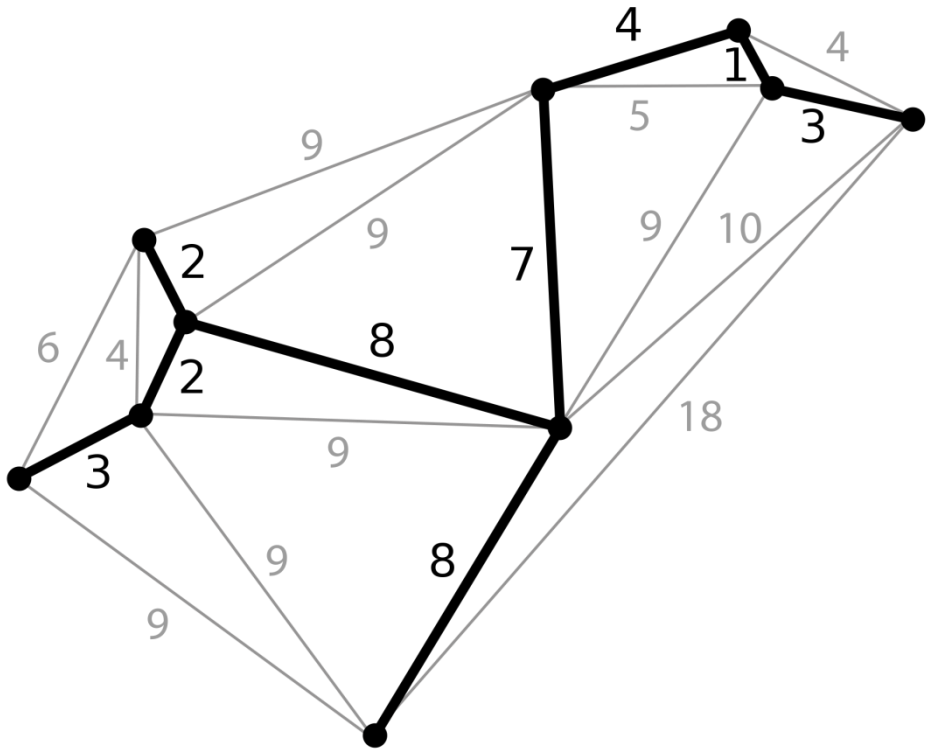
- При использовании большого числа вычислительных узлов особенно важна равномерная загрузка
- **Решение1:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин
- **Решение2:**

```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            if w ∈ Heavy then
                Out^H = Out^H ∪ w
            else
                send vertex, w to owner(w)
            end if
        end for
    end for
    broadcast Out^H
end function
```



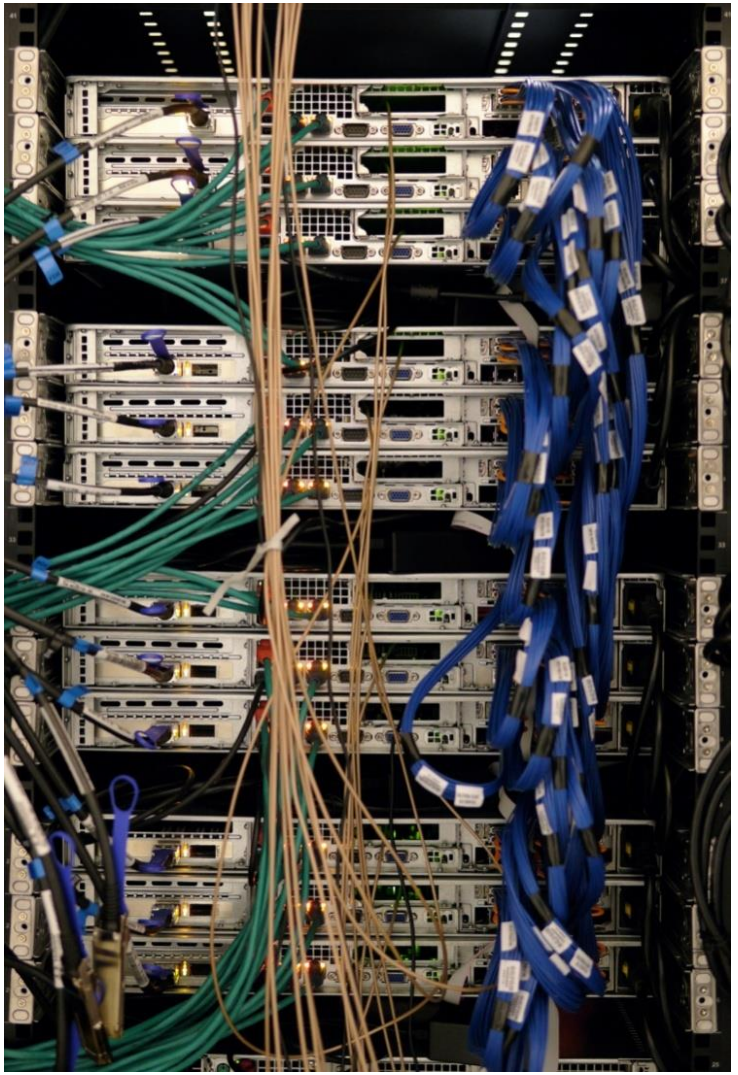
Задача поиска минимального остовного дерева (MST)

- Построение минимального остовного дерева в неориентированном графе с весами, значения которых являются действительными неотрицательными числами
- Алгоритм Gallagher, Humblet, Spira
- Короткие сообщения (30 байт)



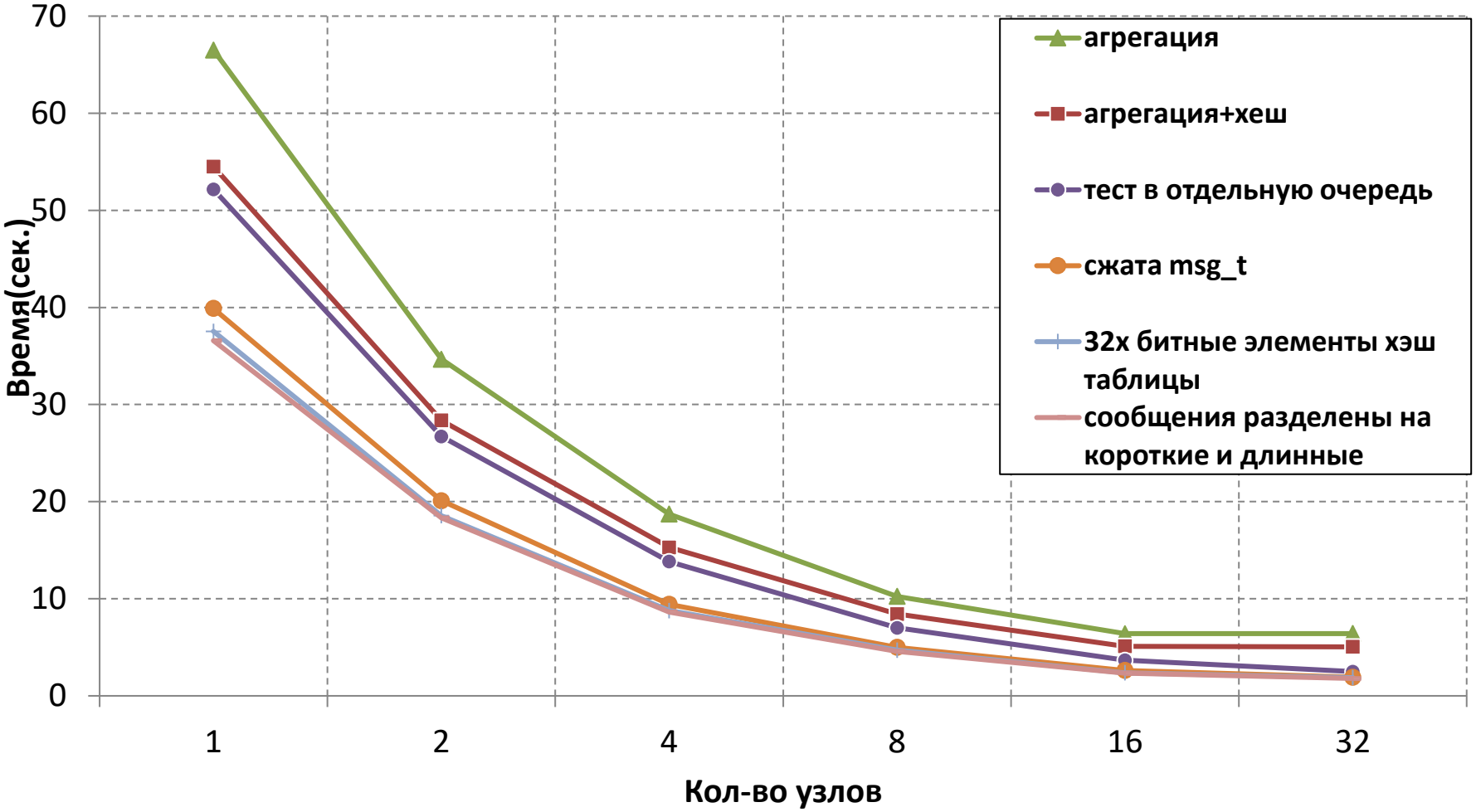
Кластер с сетью Ангара

- **24 вычислительных узла**
  - Supermicro SuperServer 5017GR-TF
  - 2 процессора Intel Xeon E5-2630 (LGA2011, 6 ядер, 2.3 ГГц)
  - 64 ГБ
- **12 вычислительных узлов**
  - Supermicro SuperServer 5017GR-TF
  - процессор Intel Xeon E5-2660 (LGA2011, 8 ядер, 2.2 ГГц)
  - 64 ГБ
- **Сеть Ангара**
  - Адаптер EC8430, топология 3D-тор 3x3x4
  - Собственная реализация SHMEM
  - MPI: MPICH 3.0.4, MVAPICH2 1.9
- **Операционная система**
  - SLES 11 SP2, Linux 3.0.13-0.27-default
  - GCC 4.3.4 (revision 152973)



Задача поиска минимального остовного дерева (MST)

Алгоритм Gallagher, Humblet, Spira. Сеть Ангара



Граф RMAT-23, средняя связность - 32

Проблемы и подходы к решению задач на распределенной памяти

- Выбор распределения данных
- Агрегация сообщений
- Организация внутриузлового параллелизма
- Уменьшение количества пересылаемых данных
- Балансировка нагрузки
- Использование эффективных коммуникаций
- Аккуратно использовать MPI
- Алгоритмические оптимизации