Параллельная обработка больших графов

Занятие 2

А.С. Семенов

dislab.org

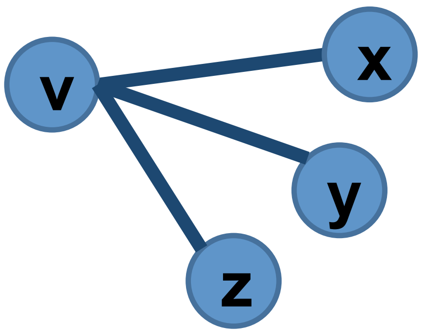
Основные проблемы, возникающие при решении задач обработки графов

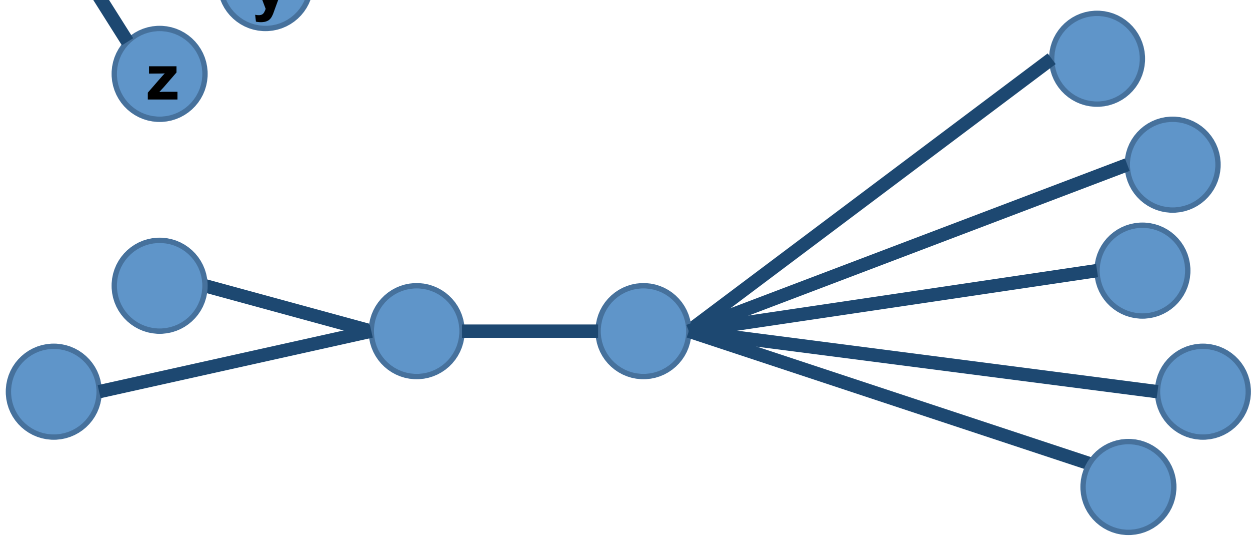
Проблемы анализа больших графов

- **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.

Проблемы анализа больших графов (1)

- **Data-driven computations.** Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.

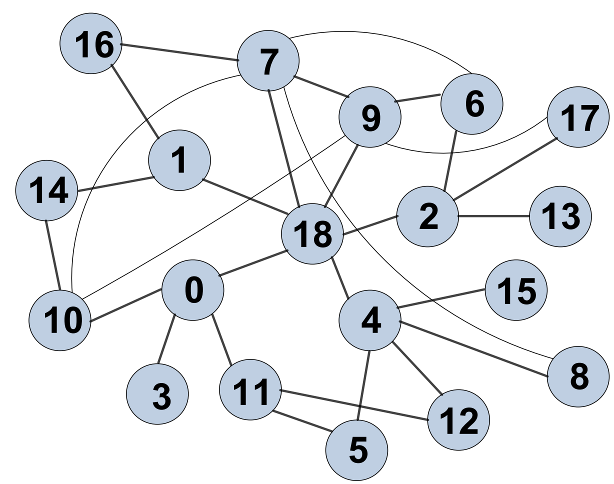




Проблемы анализа больших графов (2)

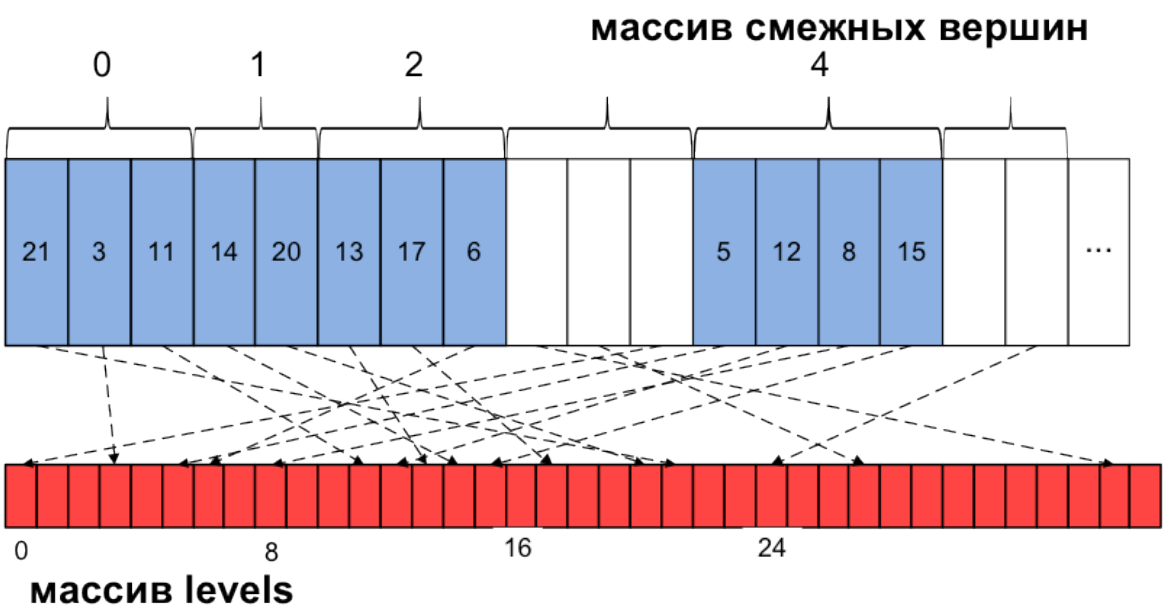
- **Unstructured problems.** Работа с нерегулярными, неструктурированными данными, трудность распараллеливания.

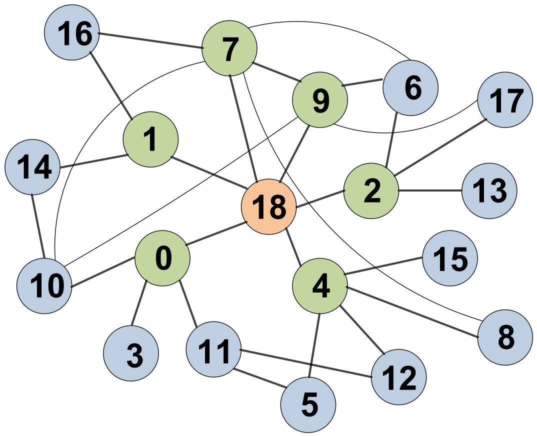
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1				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



Проблемы анализа больших графов (3)

- **Poor locality.** Низкая пространственно-временная локализация обращений к памяти.





Проблемы анализа больших графов (4)

- **High data access to computation ratio.** Преобладание команд доступа к памяти над командами выполнения арифметических операций.

Intel E5-2680 v3, 2.5 ГГц

Параметр	**Задержка, нс (такты)**	**ПС, ГБ/с**
Регистр	(1)	–
Кэш L1	1.6 (4)	240
Кэш L2	4.4 (11)	160
Кэш L3	16 (40)	80
Память своего сокета	60	~55
Память чужого сокета	100	~30
Сеть Ангара	MPI – 1000 нс, SHMEM – 600 нс	

Алгоритмы обработки графов

Поиск вширь в графе (Breadth-first Search, BFS)

BFS (G, v_{start}) $\qquad G = (V, E)$

$Q = \{v_{start}\}$

$Visited = \{v_{start}\}$

while $Q \neq \{\}$

$\quad Q_{next} = \{\}$

$\quad$ **for all** vertex $\in Q$ **do**

$\quad\quad$ **for all** w: (vertex, w) $\in E$ **do**

$\quad\quad\quad$ **if** $w \notin Visited$ **then**

$\quad\quad\quad\quad Q_{next} = Q_{next} \cup w$

$\quad\quad\quad\quad Visited = Visited \cup w$

$\quad\quad\quad$ **endif**

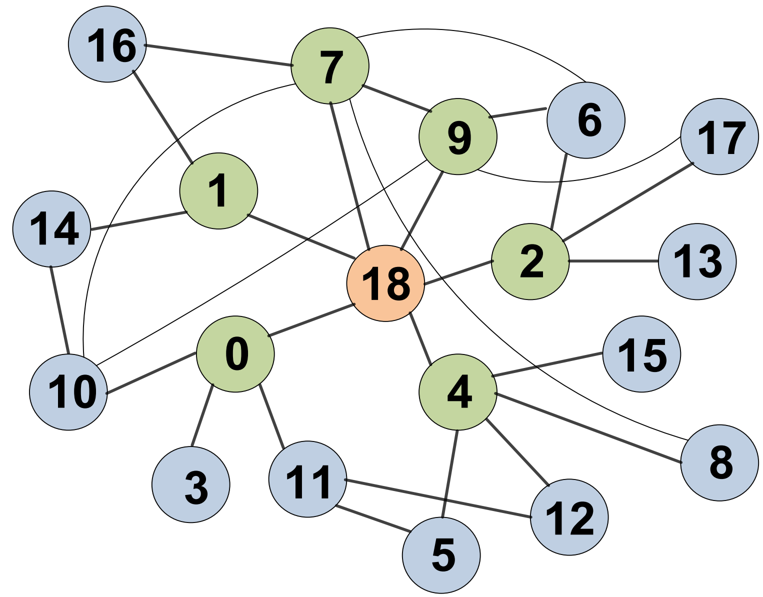
$\quad\quad$ **end for**

$\quad$ **end for**

$\quad Q = Q_{next}$

end while

Сложность **O(N+M)**



Поиск вширь в графе (Breadth-first Search, BFS)

<u>BFS</u> (G, v_{start}) $\qquad$ $G = (V, E)$

$Q = \{v_{start}\}$

$Visited = \{v_{start}\}$

while $Q \neq \{\}$

  $Q_{next} = \{\}$

  #pragma omp parallel for

  for all vertex $\in Q$ **do**

    for all w: (vertex, w) $\in E$ **do**

      if $w \notin Visited$ **then**

        #pragma omp critical

          $Q_{next} = Q_{next} \cup w$

          $Visited = Visited \cup w$

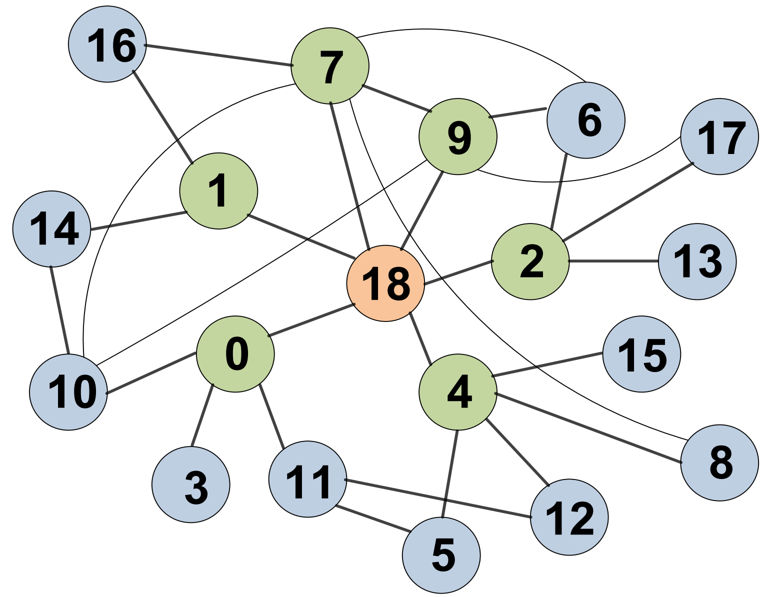
      endif

    end for

  end for

  $Q = Q_{next}$

end while

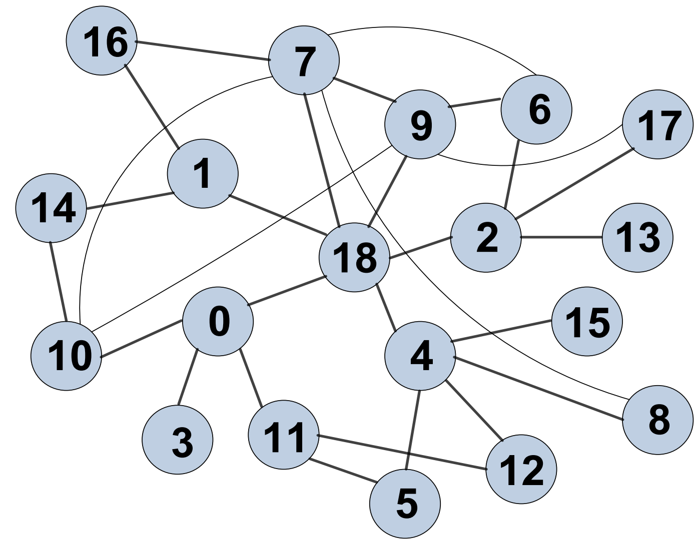


Поиск вглубь в графе
(Depth-first Search, DFS)

<u>**StartDFS**</u> (G, v_{start})   G=(V, E)
Visited = $\{v_{start}\}$
DFS(G, v_{start})

<u>**DFS**</u> (G, vertex)
Visited = Visited ∪ vertex
for all w: (vertex, w) ∈ E **do**
    if w ∉ Visited **then**
        DFS(G, w)
    endif
end for

Сложность **O(N+M)**



Определения

- **Дерево** – связный граф без циклов
- **Остовное дерево связного графа** – подграф, являющийся деревом и связывающий все вершины исходного графа
- **Лес** – граф, каждая связная компонента которого является деревом
- **Остовный лес для графа G** – граф, являющийся объединением остовных деревьев всех компонент связности G

Задача построения минимального остовного дерева (Minimum Spanning Tree, MST)

- **Минимальный остовный лес графа G** – остовный лес, вес которого не превосходит вес любого другого возможного остовного дерева графа G
- **Minimum Spanning Tree**
  – Дан неориентированный граф G = (V, E, W)
  – W: E → R[0;1] – весовая функция
  – Найти минимальный остовный лес графа G

Алгоритмы решения задачи MST

- Прима, 1957
- Крускала, 1950/1930
- Борувки, 1926
- GHS (Gallager, Humblet, Spira), 1983

MST, жадные алгоритмы

```
MST (G, w)
A = {} // множество ребер
while A не является остовным деревом
    (u, v) = argmin W[(u, v)],
        u∈ ребру из A, v ∉ ни одному ребру из A
    A = A U {(u, v)}
end while
```

MST, алгоритм Прима

```
MST-Prim (G, w, r) // G – связный граф
for all u∈V do u.key=inf; u.par=undef end for
key[r] = 0
Q = V
A = {}
while Q <> {}
        u = ExtractMin(Q)
        A = A U {(u.par, u)}
        for all v∈Adj[u]
            if v∈Q and w(u,v) < key[v]
                v.key=w(u,v); v.par = u
                // с вызовом DecreaseKey(Q, v, w(u, v))
            end if
        end for
end while
```

Сложность **O(M lgN), O(M+N lgN)**

MST, алгоритм Крускала

```
MST-Kruskal (G, w, r) // G – связный граф
A = {}
for all u∈V do MakeSet(u) end for
Sort(E)
for all (u,v)∈E
    if FindSet(u) != FindSet(v)
        A = A U {(u, v)}
        Union(u, v)
    end if
end for
```

Система непересекающихся множеств

Также известна как Union-Find

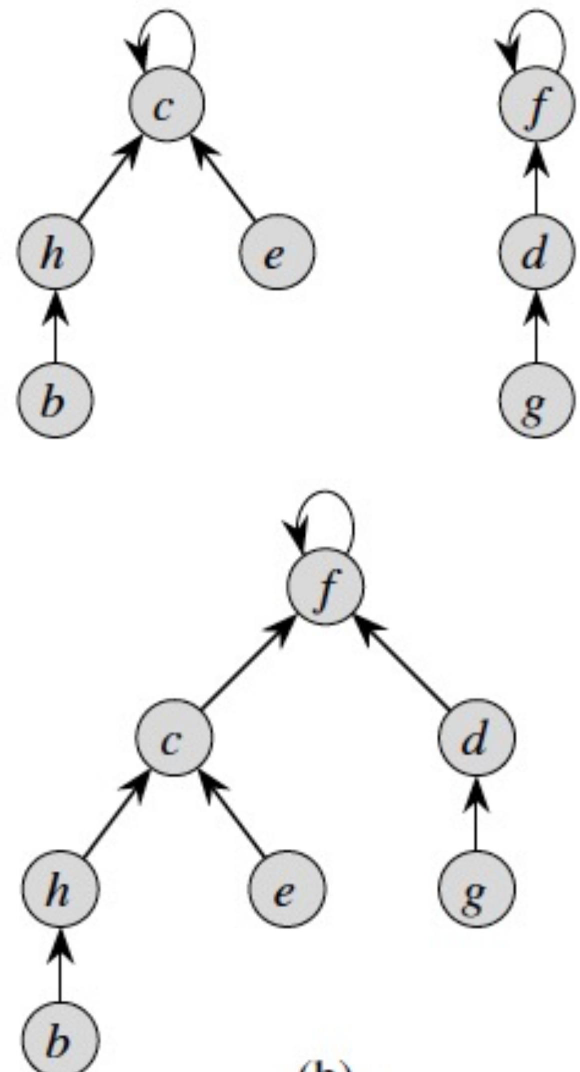
1. **MakeSet(v)** создать отдельное множество из элемента v
2. **FindSet(v)** → **u** найти элемент-представитель множества, содержащего вершину v
3. **Union(u,v)** объединить множества, содержащие элементы u и v

Реализация Union-Find

<u>MakeSet</u> (v)
v.parent = v

<u>FindSet</u> (v)
if v.parent == v
  return v
end if
return FindSet(v.parent)

<u>Union</u> (u, v)
x = FindSet(u)
y = FindSet(v)
x.parent = y



(b)

Оптимизация Union-Find

```
MakeSet (v)
v.parent = v
v.rank = 0 // rank – верхняя граница высоты дерева

FindSet (v):
if v != v.parent
    v.parent = FindSet(v.parent)
end if
return v.parent
```

Оптимизация Union-Find

```
Union (u,v)
Link(FindSet(u), FindSet(v))

Link (x, y) // x, y - корни
if x.rank > y.rank
    y.parent = x
else
    x.parent = y
    if x.rank == y.rank
        y.rank = y.rank + 1
    end if
end if
```

Сложность **$O(m\ \alpha(N)) \le O(4m)$,**
m – количество операций

MST, алгоритм Крускала

```
MST-Kruskal (G, w, r) // G – связный граф
A = {}
for all u∈V do MakeSet(u) end for
Sort(E)
for all (u,v)∈E
    if FindSet(u) != FindSet(v)
        A = A U {(u, v)}
        Union(u, v)
    end if
end for
```

Сложность **O(M logN)**

MST, алгоритм Борувки

<u>MST-Boruvka</u> (G, w) // G – связный, различные веса
T = { MakeSet(v_0), MakeSet(v_1), …}
A = {}
while |T| > 1
    L = {}
    for all component $C_i \in T$
        (u, v) = argmin W[(u, v)], $u \in C_i$, $v \notin C_i$
        L = L U {(u, v)}
    end for
    for all (u, v) $\in L$
        if $u \in C_i$, $v \notin C_i$ **then**
            Union(u, v)
            A = A U {(u, v)}
    end for
end while

Сложность **O(M log N)**

MST, алгоритм Борувки

```
MST-Boruvka (G, w) // G – связный, различные веса
T = { MakeSet(v0), MakeSet(v1), …}
A = {}
while |T| > 1
      L = {}
      #pragma omp parallel for
      for all component Ci ∈T
            (u, v) = argmin W[(u, v)], u∈ Ci, v ∉ Ci
            L = L U {(u, v)}
      end for
      #pragma omp parallel for
      for all (u, v) ∈L
            if u∈ Ci, v ∉ Ci then
                  Union(u, v)
                  A = A U {(u, v)}
      end for
end while
```