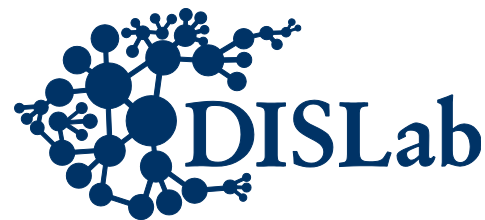


Параллельная обработка больших графов

Александр Сергеевич Семенов

dislab.org



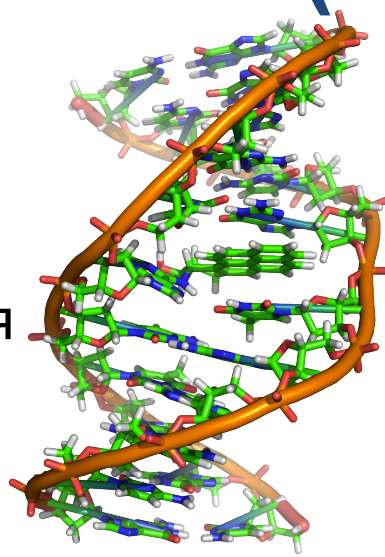
Откуда возникают большие графы?

- **Интернет (WWW)**
 - На сентябрь 2016 – 47 миллиардов страниц
 - По оценке Google – более 1 триллиона
- **Социальные медиа**
 - Блогосфера: 2011 – 172×10^6 (+ 10^6 /день)
 - Facebook: 2010 – 500×10^6 , 2013 – 1.1×10^9 (650×10^6 акт.польз./день), 140×10^9 связей
 - LinkedIn: 2013 – 8×10^6 , 60×10^6 связей
 - Twitter: 2011 – 140×10^6 сообщений/день
- **Транспортные сети**
- **Биоинформатика**
- **Бизнес-задачи**

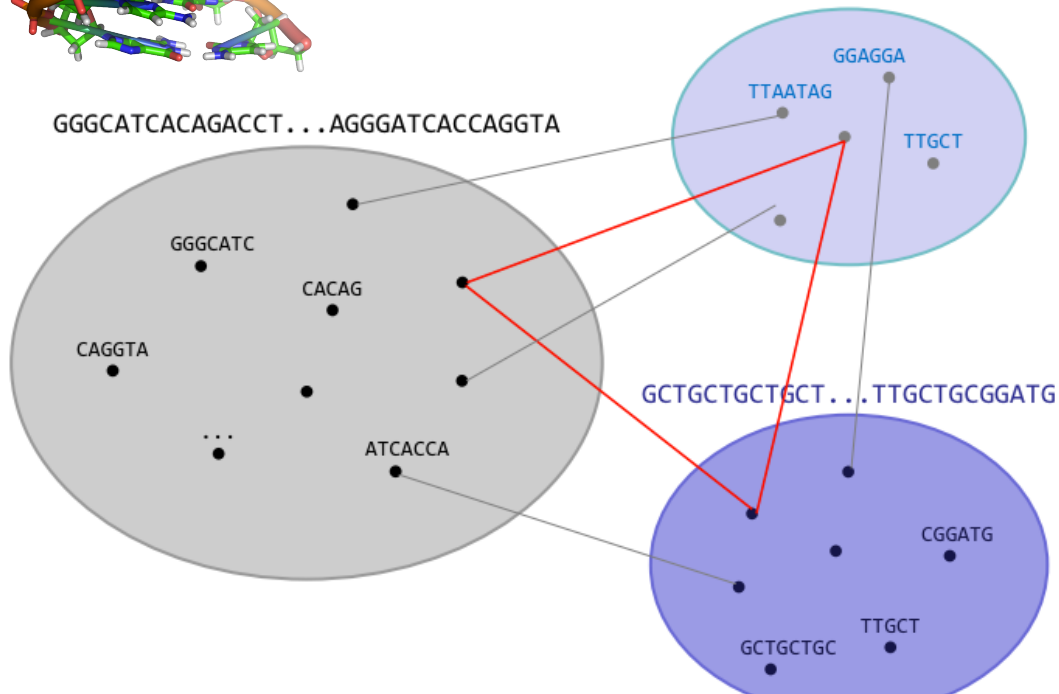
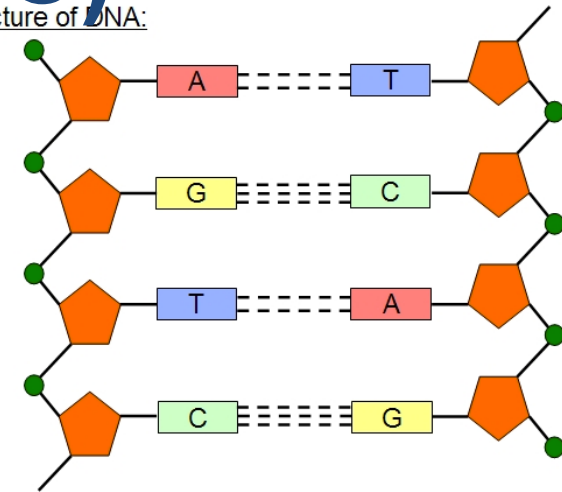
¹<http://www.worldwidewebsize.com>

Биоинформатика: сходство организмов (НРС)

- Число долей 10^5
- Длина последовательности 10^9
- Вершин в доле 10^9 (берутся короткие слова)
- Всего вершин 10^{14}
- Найти слова, которые с заданной точностью встречаются во всех последовательностях, или
- Найти клику или плотный подграф (кластеризация), если ребро – характеристика сходства

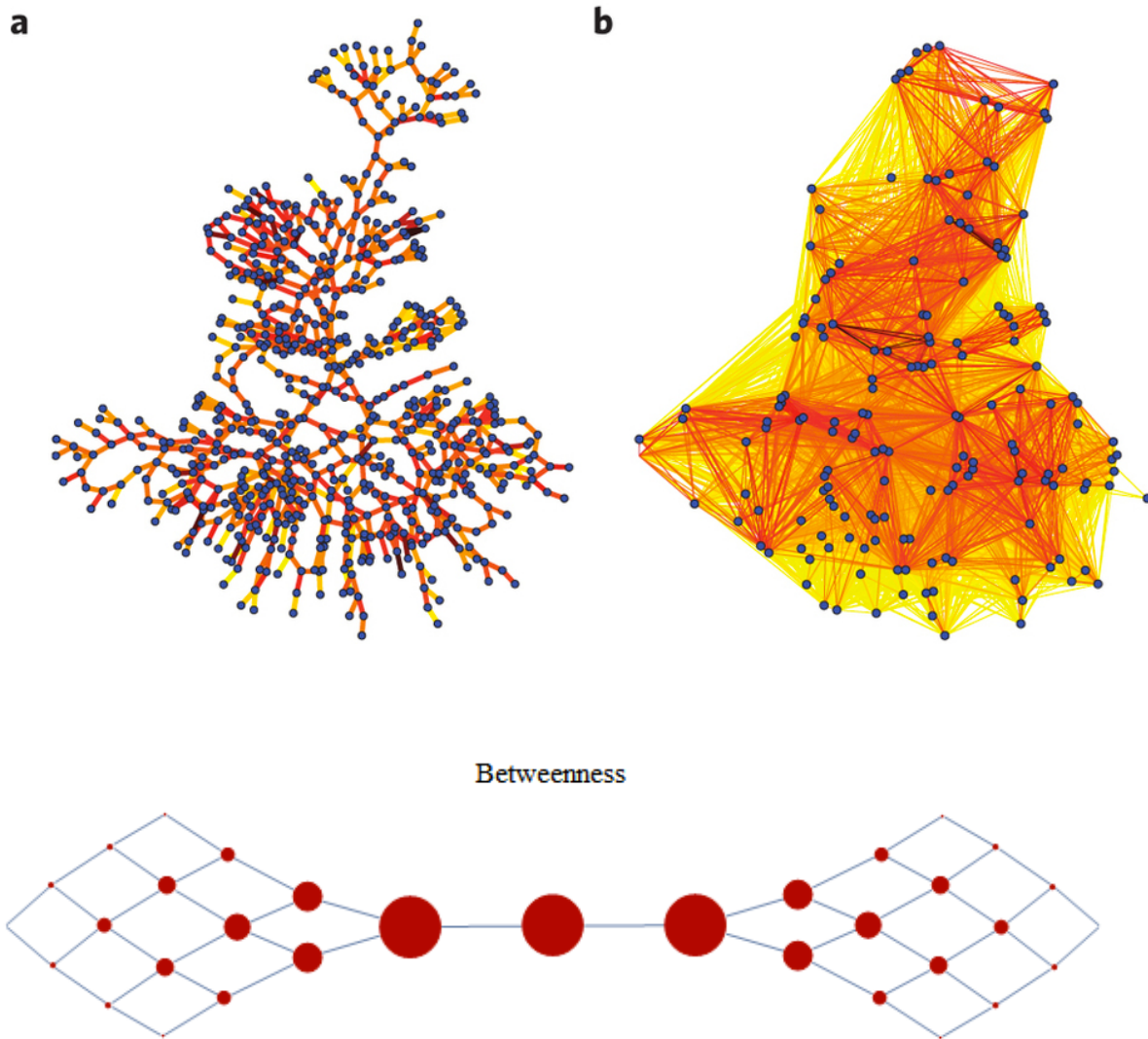


Structure of DNA:



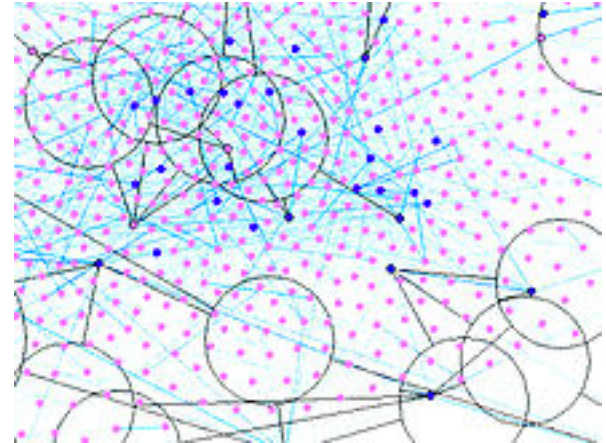
Электросети (НРС)

- Связанность
- Надежность
- Различные пути,
betweenness
centrality



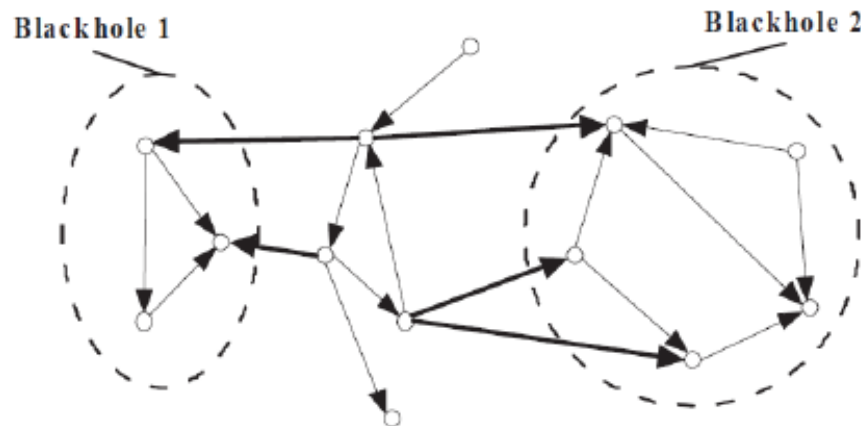
Анализ социальных сетей (НРС)

- Анализ сообществ
- Понимание намерений
- Динамика популяции
- Распространение эпидемий
- Кластеризация



Бизнес-аналитика и кибербезопасность (Big Data&HPC)

- Задачи понимания данных из огромных массивов
- Выявление аномалий в данных
- Анализ данных
- Выявление мошенничества
- Паттерн «черные дыры»
- Machine Learning!



Признаки в графах для машинного обучения

- Вершины (степень, полустепени, betweenness centrality, PageRank)
- Пары вершин (количество общих соседей, вес ребра)
- Egonet (количество треугольников, количество ребер)
- Группа вершин (плотность = кол-во ребер/кол-во вершин, общий вес ребер)

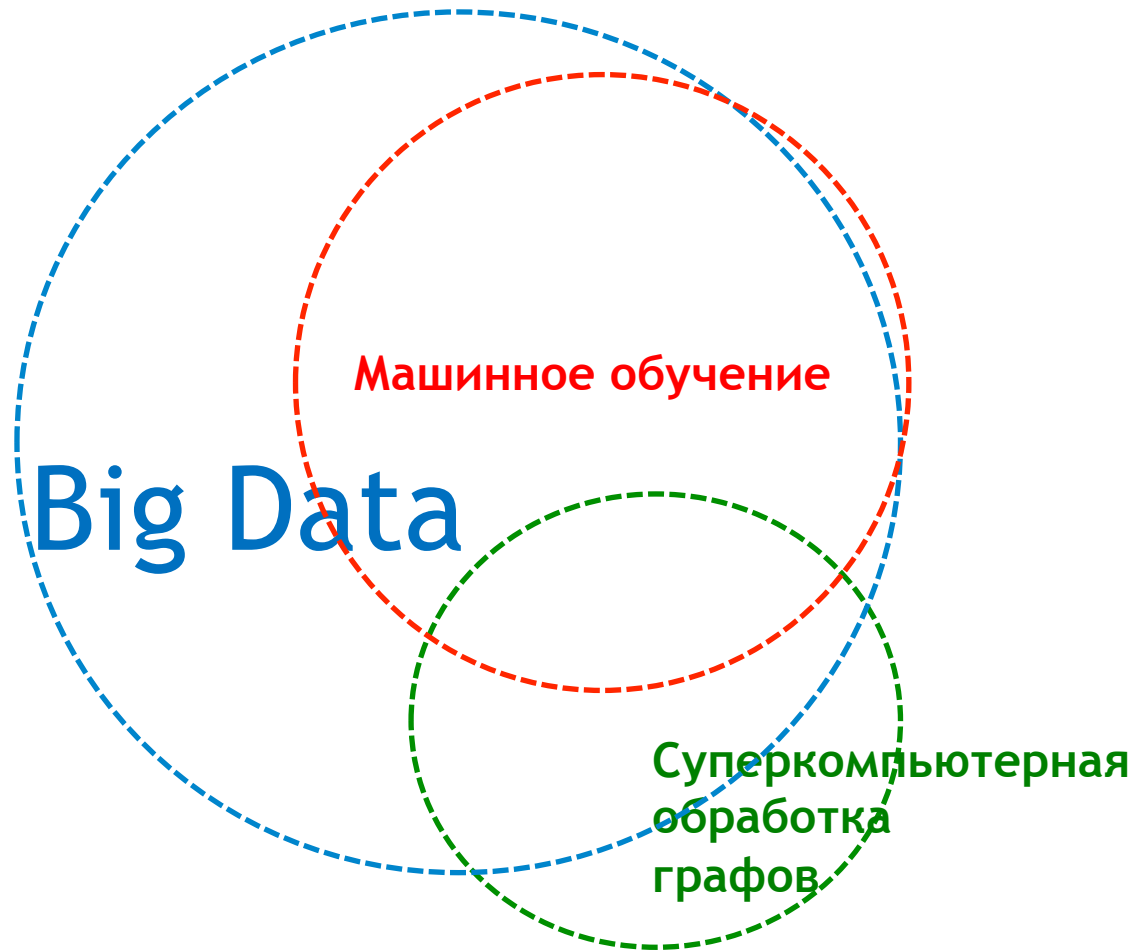
Классификация задач анализа графов

- По типу графов
 - статические графы (static graph analysis)
 - динамические графы (dynamic graph analysis)
 - обработка потоков вершин и ребер (streaming graph analysis)
- По типу обработки
 - в режиме реального времени (online)
 - в режиме выполнения заданий (offline, batch processing)

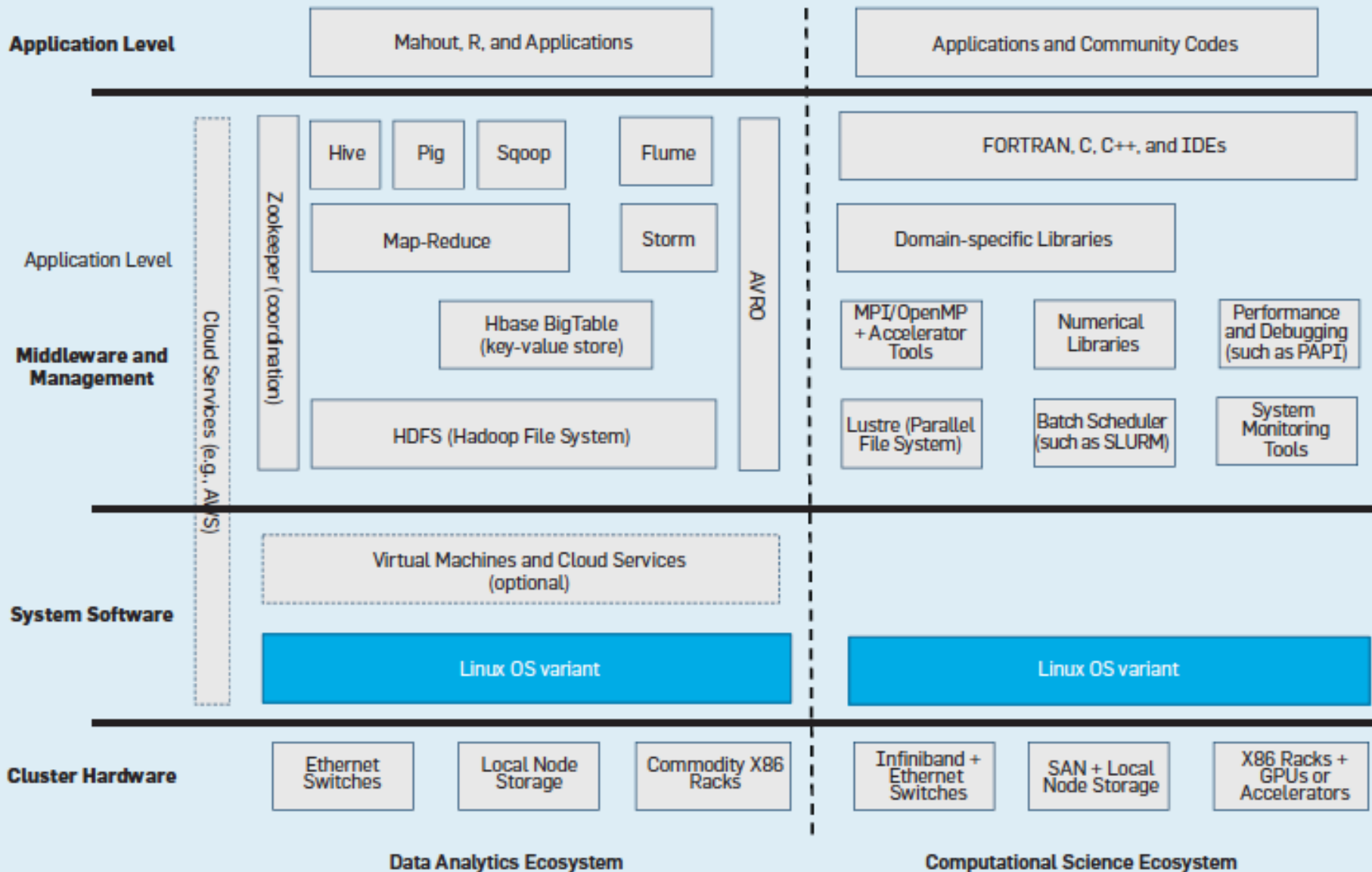
Программные модели и средства

- Реляционная модель
 - Cassandra, SAP HANA, ...
- MapReduce
- Generic MR:
 - Hadoop, Yarn, Dryad, Stratosphere, Haloop
- Graph-optimized: Pegasus, Surfer, GBASE, GraphX
- Специализированные языки программирования
 - Проблемно-ориентированные языки программирования (DSL)
 - Green-Marl, Exedra
 - Языки запросов к графовым СУБД
 - SPARQL, G-SPARQL, Cypher (Neo4j), ...
- BSP
 - Parallel BGL
- Vertex-centric/BSP
 - Pregel (Giraph, Hama, Mizan, ...)
- Vertex-centric/Data, Message-driven
 - GraphLab, SWARM, Trinity, Charm++, ...
- Fine-grained Threaded Shared Memory/PGAS
 - GraphCT, STINGER, Grappa
- Технологии параллельного программирования
 - OpenMP, MPI, CUDA, ...

Big Data vs HPC



Big Data vs HPC



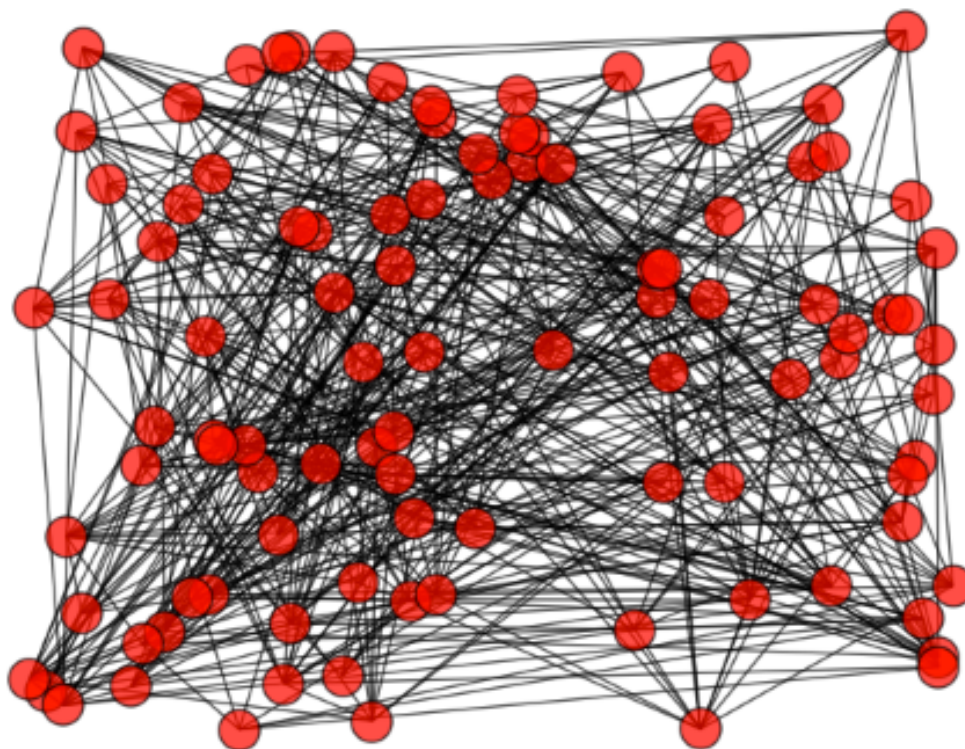
План

- Виды графов
- Основные проблемы, возникающие при решении задач обработки графов
- Подходы к решению задач в рамках одного вычислительного узла
- Подходы к решению задач в рамках распределенной вычислительной системы
- Сеть Ангара

Виды графов

Виды графов. Случайные графы

- Random, Random Uniform, Erdos Renyi
- N вершин, M ребер, k – средняя связность вершины



Виды графов. Степенной закон

- WWW, Социальные сети, Биоинформатика

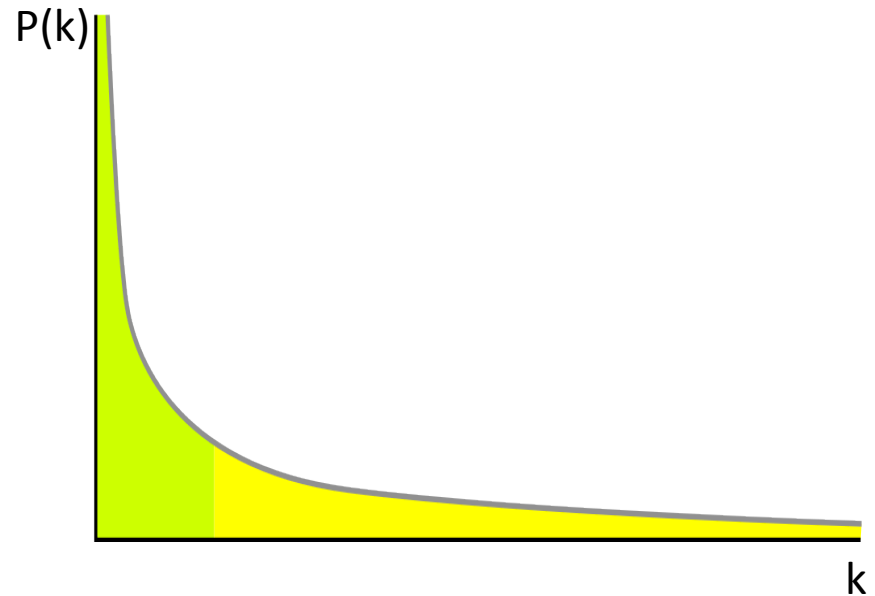
- Графы small-world

$L \sim \log N$

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

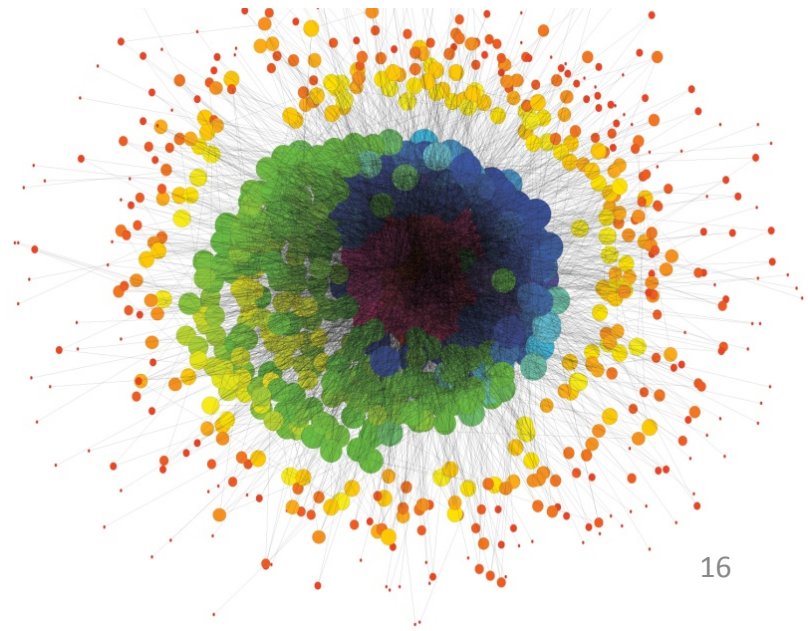
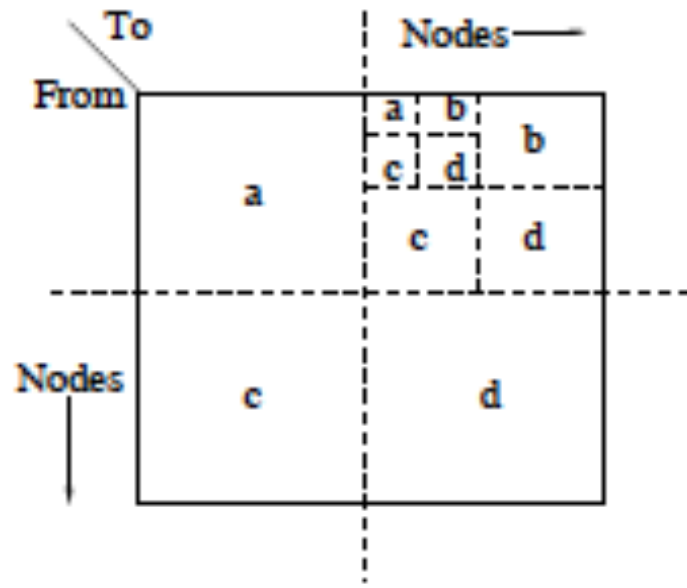
k – связность вершины

$L \sim \log \log N$



Виды графов. RМAТ-граф

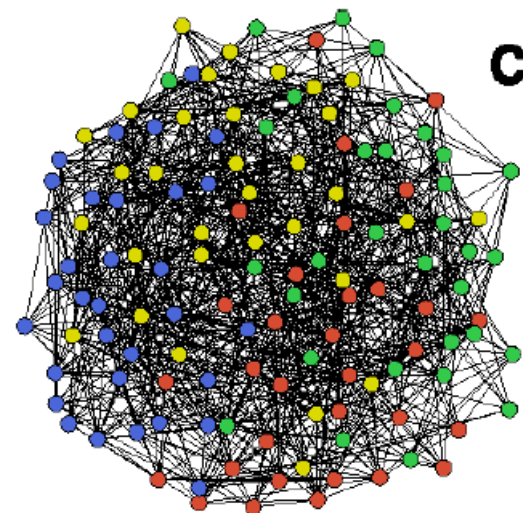
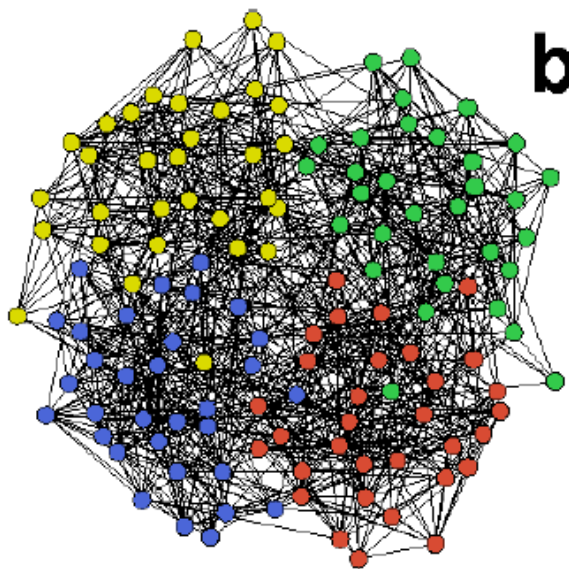
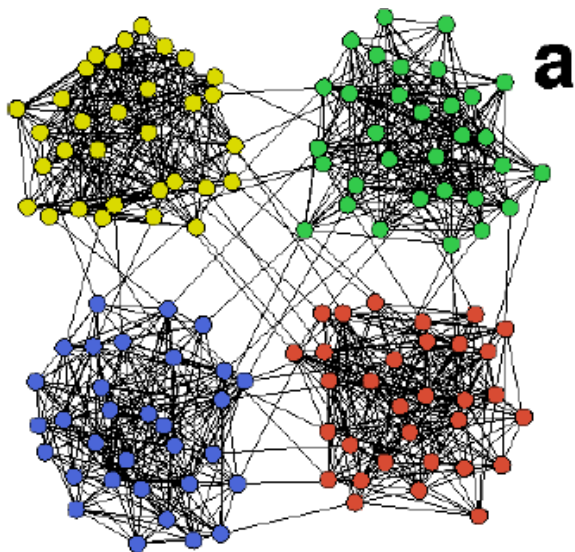
- $a+b+c+d = 1$
- Сообщества:
 - a и d – сообщества
 - b и c – связи между ними
 - наличие «подсообществ»
- может быть scale-free при $a \geq d$
- случайная перестановка вершин



Виды графов. LFR*-граф

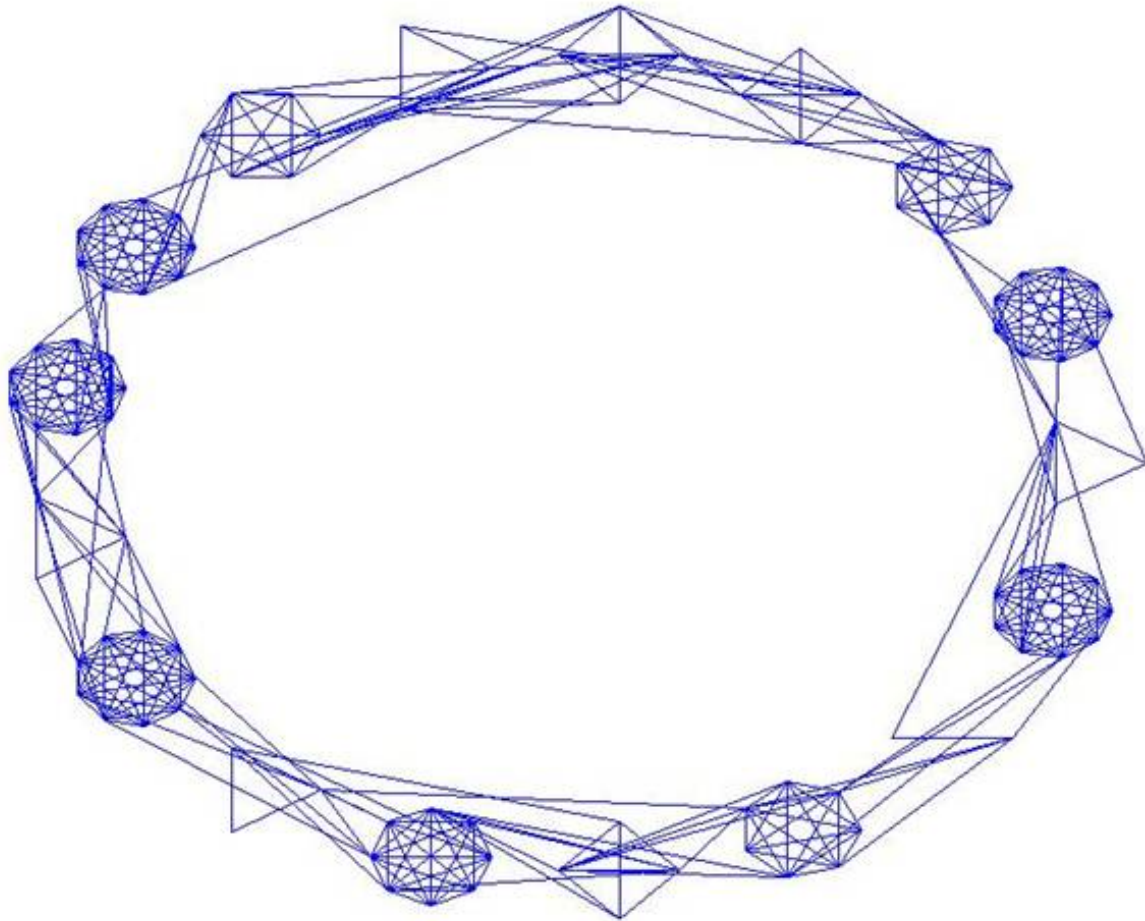
- **Параметры:**

- $m_i \in [0;1]$, показывает количество связей вне сообщества
- com_tau – показатель степени в законе распределения размеров сообществ
- deg_tau – показатель степени в законе распределения степеней вершин



Виды графов. SSCA2-граф

- Равномерное распределение случайных параметров
- случайная перестановка вершин



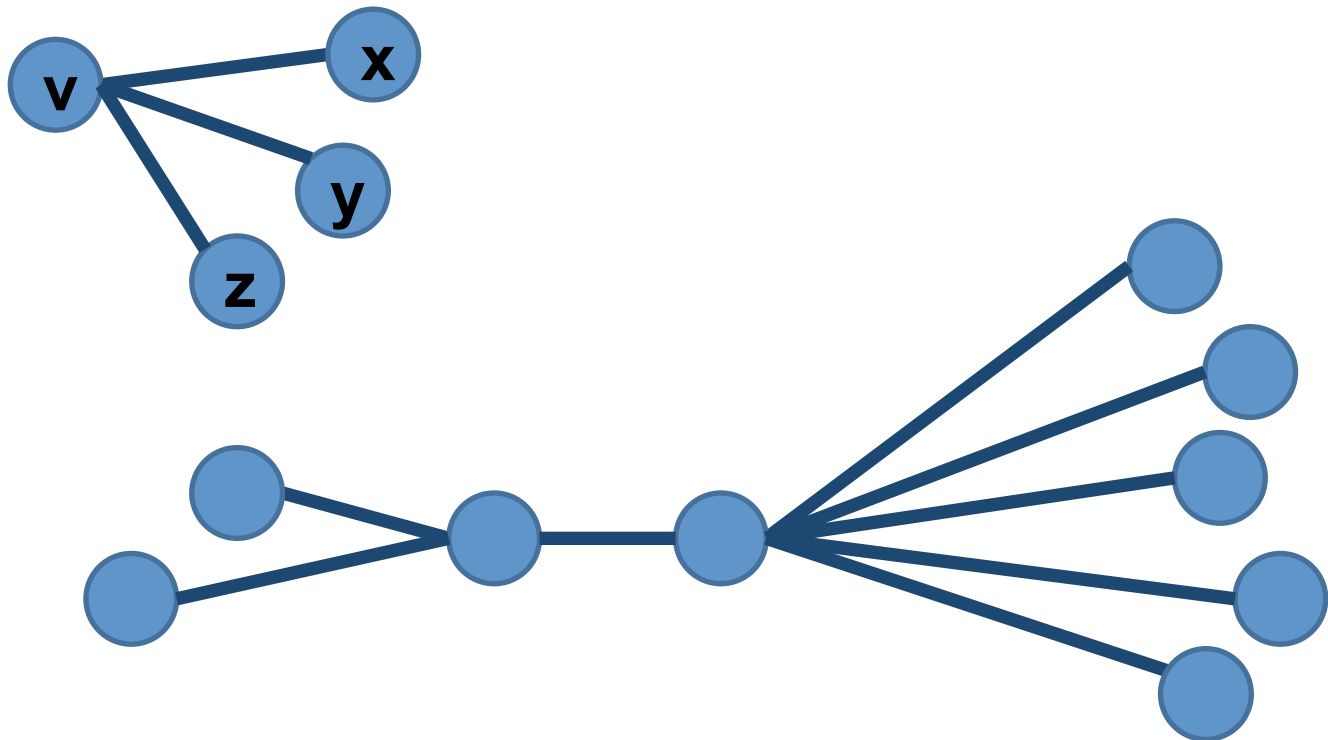
Основные проблемы, возникающие при решении задач обработки графов

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)

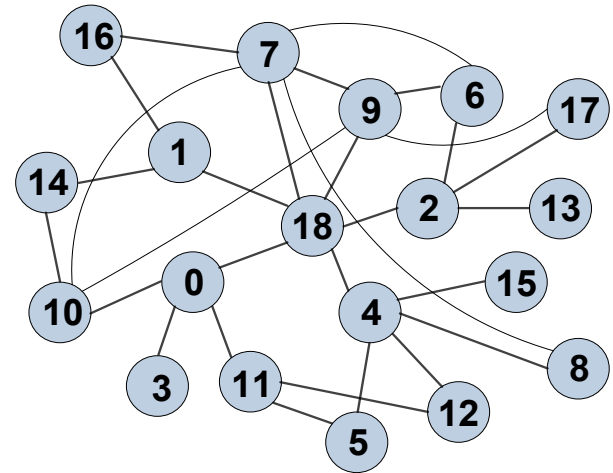
- **Data-driven computations.** Зависимость вычислений от данных (топологии графа). Невозможность применения методов статического распараллеливания вычислений.



Проблемы анализа больших графов (2)

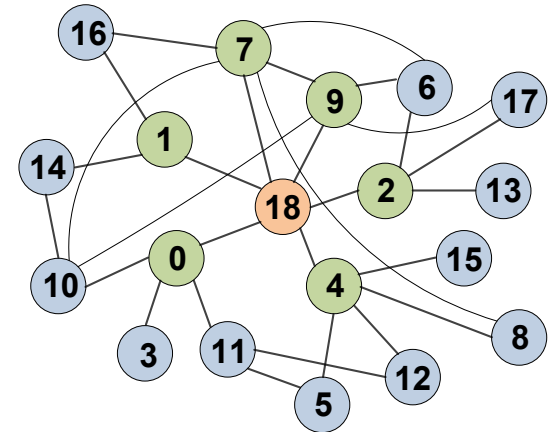
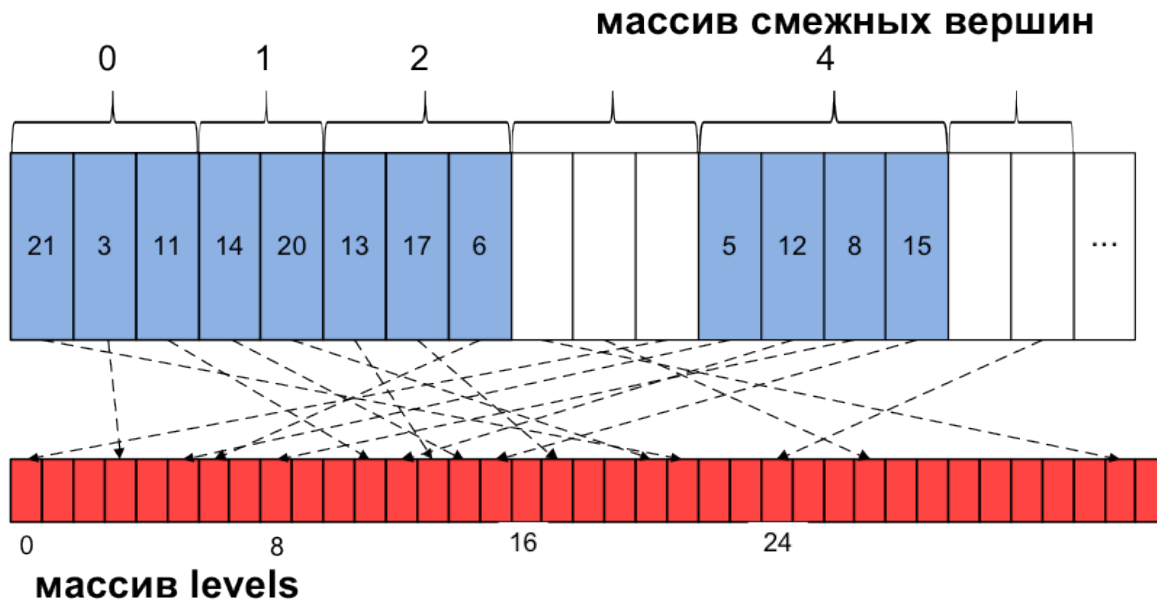
- **Unstructured problems.** Работа с нерегулярными, неструктурированными данными, трудность распараллеливания.

	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)

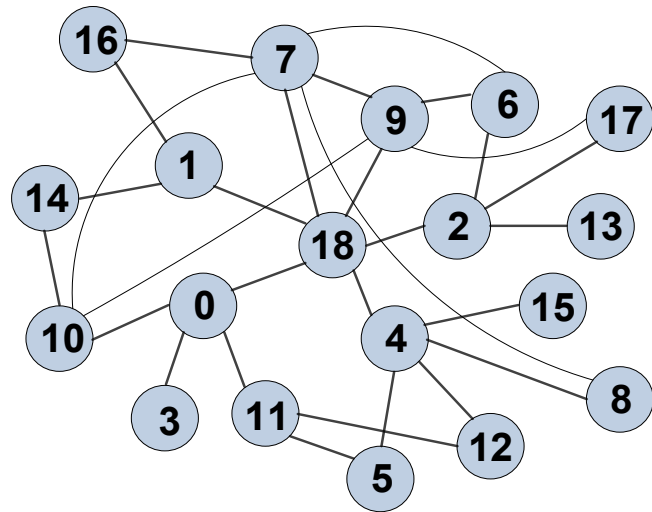
- **High data access to computation ratio.** Преобладание команд доступа к памяти над командами выполнения арифметических операций.

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	

Проблемы и подходы к решению задач обработки графов в рамках одного вычислительного узла

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									

Форматы представления разреженных матриц

- Доля ненулевых элементов мала

Можно хранить только позиции и значения ненулевых элементов

- Compressed Row Storage (CRS)
- Coordinate list (COO)
- DIA
- ELLPACK
- SELLPACK
- Оптимизированный под задачу

Внутреннее представление Compressed Row Storage (CRS)

Sparse Matrix

10	0	0	0	-2
3	9	0	0	0
0	7	8	7	0
3	0	8	7	5
0	8	0	9	13

Row pointer array

0	2	4	7	11	14
---	---	---	---	----	----

rowsIndices

Column indices array

0	4	0	1	1	2	3	0	2	3	4	1	3	4
---	---	---	---	---	---	---	---	---	---	---	---	---	---

endV

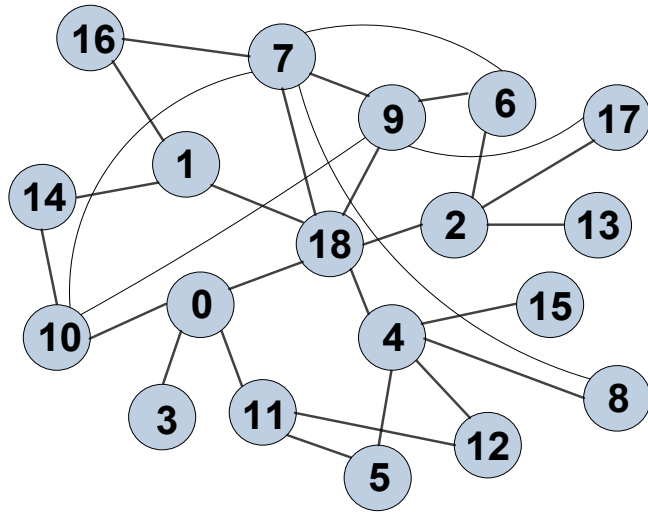
Values array

10	-2	3	9	7	8	7	3	8	7	5	8	9	13
----	----	---	---	---	---	---	---	---	---	---	---	---	----

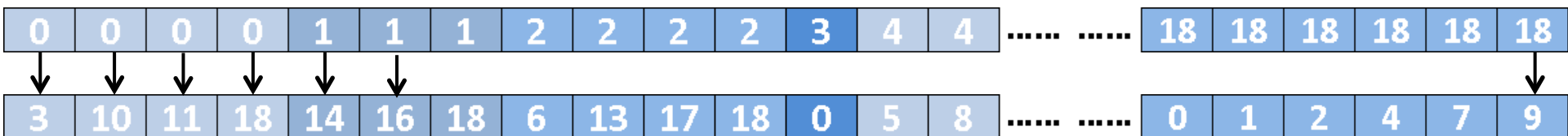
weights

```
for (int u = 0; u < G->n; u++) {  
    for (int j = G->rowsIndices[u]; j < rowsIndices[u+1];  
        j++) {  
        const int v = G->endV[j];  
        const int w = G->weights[j];  
        // обработка ребра u->v  
    }  
}
```

Coordinate list (COO)



	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									

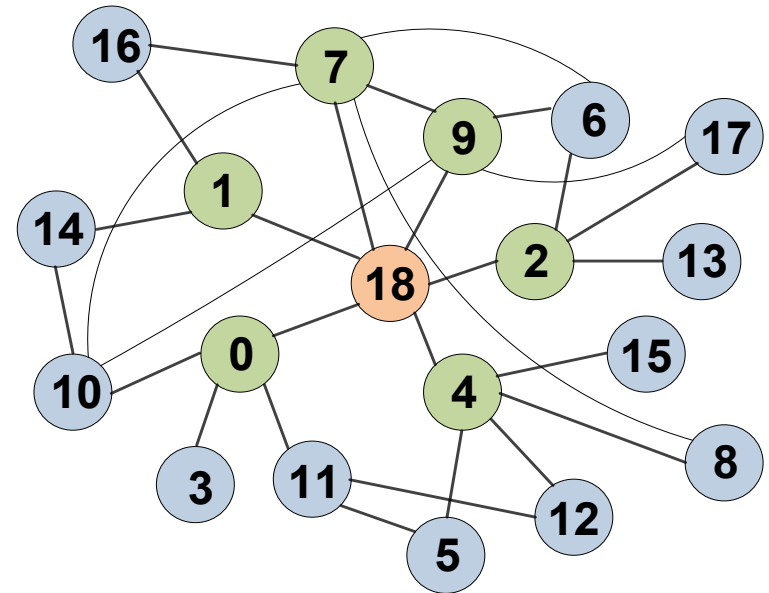


Поиск вширь в графе

```

Q = {vstart}
Visited = {vstart}
while Q ≠ {}
    Qnext = {}
    for all vertex ∈ Q do
        for all w: (vertex, w) ∈ E do
            if w ∉ Visited then
                Qnext = Qnext ∪ w
                Visited = Visited ∪ w
            endif
        end for
    end for
    Q = Qnext
end while

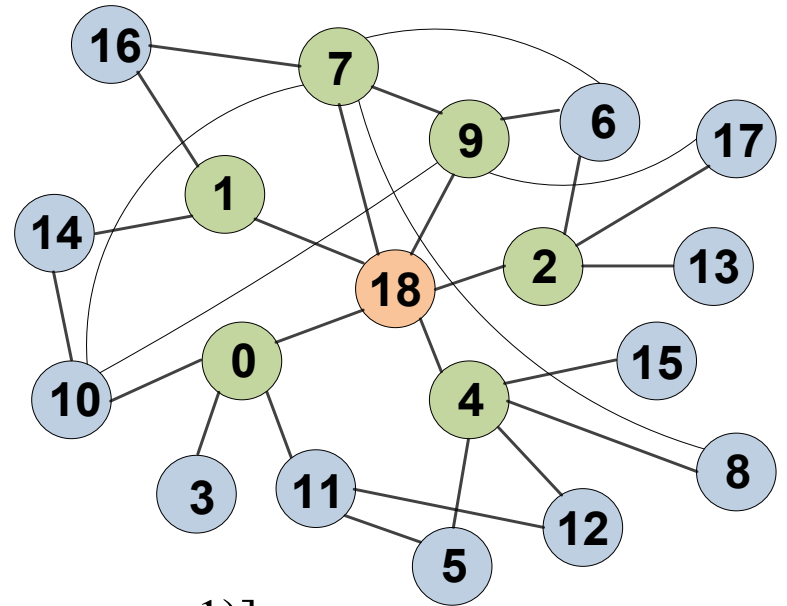
```



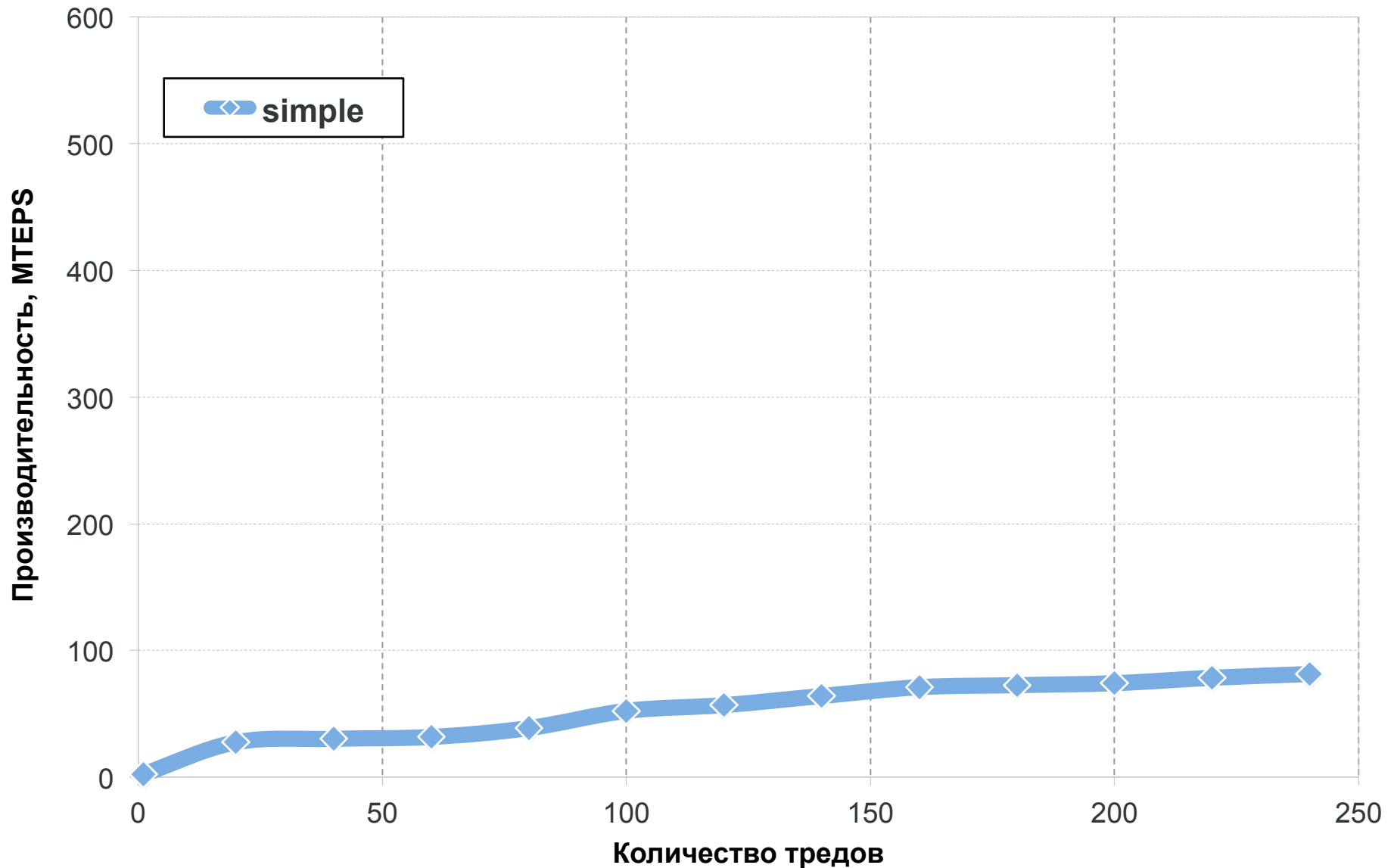
Поиск вширь в графе (BFS)

Подход Queue-based, алгоритм simple

```
Q_counter = 1
Q[0] = root
Visited[root] = 1
while Q_counter > 0
    Q_next_counter = 0
    #pragma omp parallel for
    for all vertex  $\in$  Q do
        for all w: (vertex, w)  $\in$  E do
            if Visited[w] == 0 then
                Q_next[__sync_fetch_and_add(Q_next_counter, 1)] = w
                Visited[w] = 1
            endif
        end for
    end for
    swap(Q, Q_next) // обмен Q и Q_next
end while
```

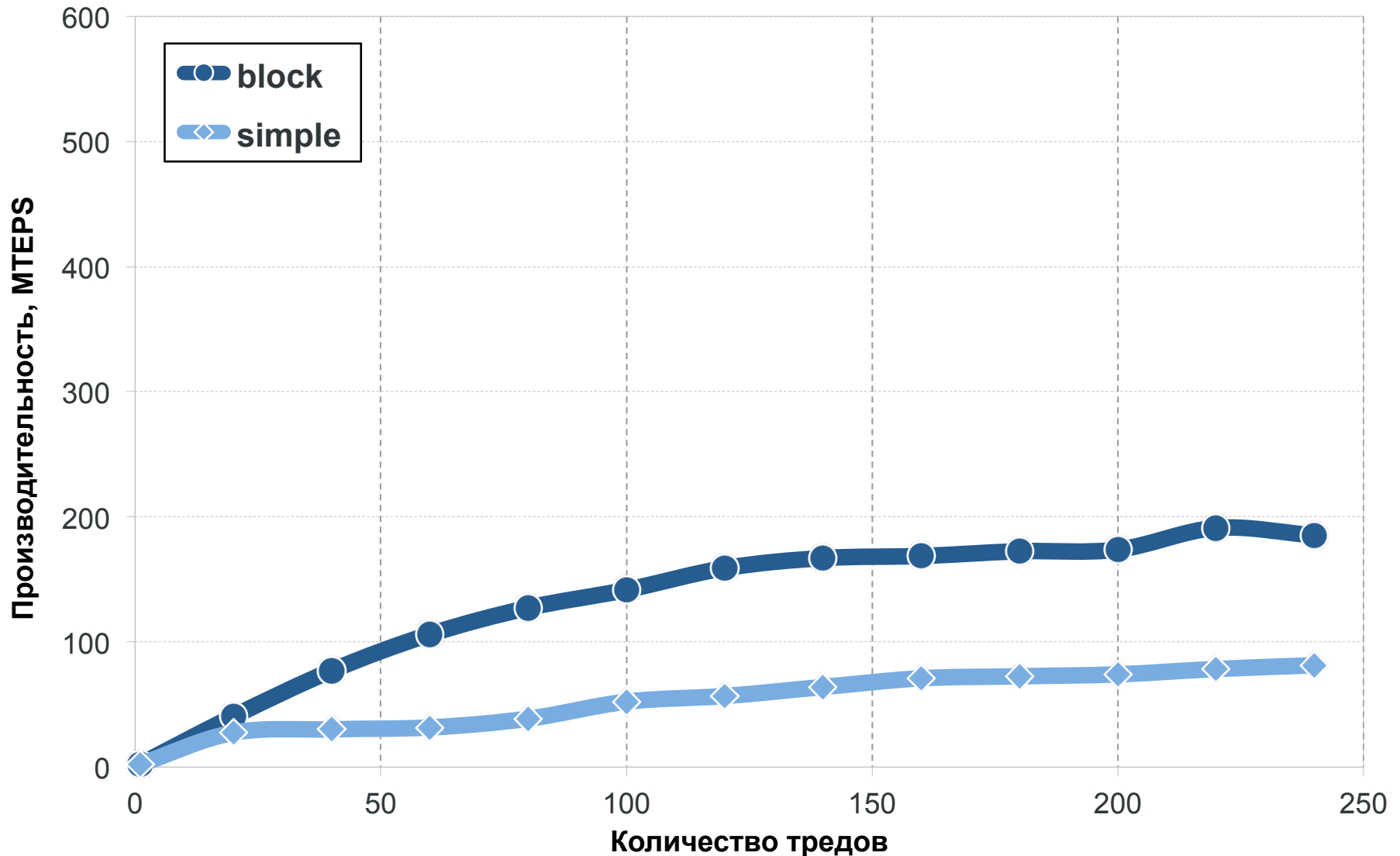


Производительность алгоритма simple в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



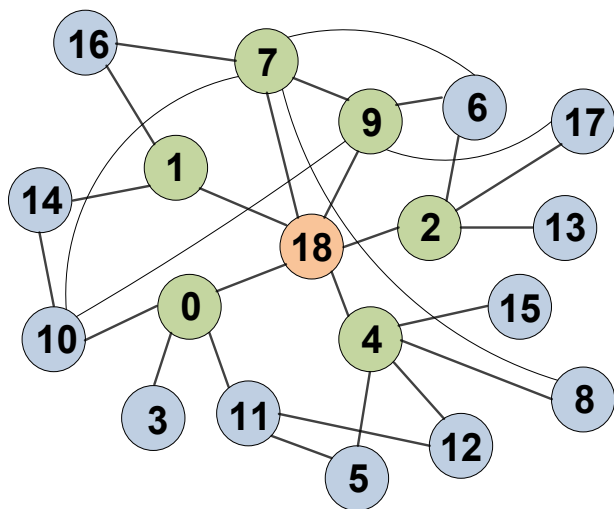
Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Производительность алгоритмов simple и block в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

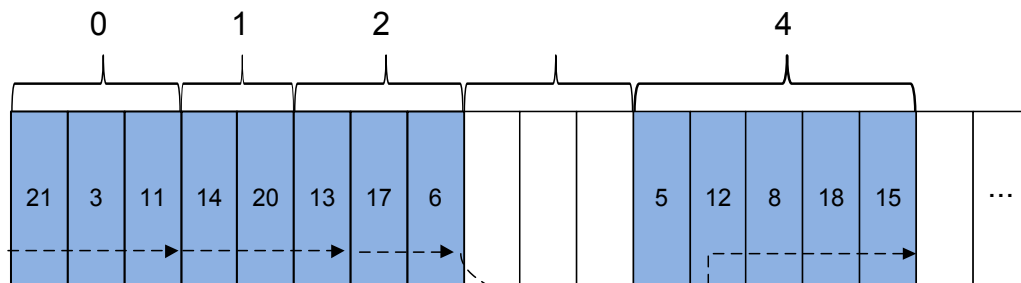


Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Подход Read-based, алгоритм read



0	1	2	3	4	массив levels				
1	1	1	INF	1	INF	INF	1	0	1

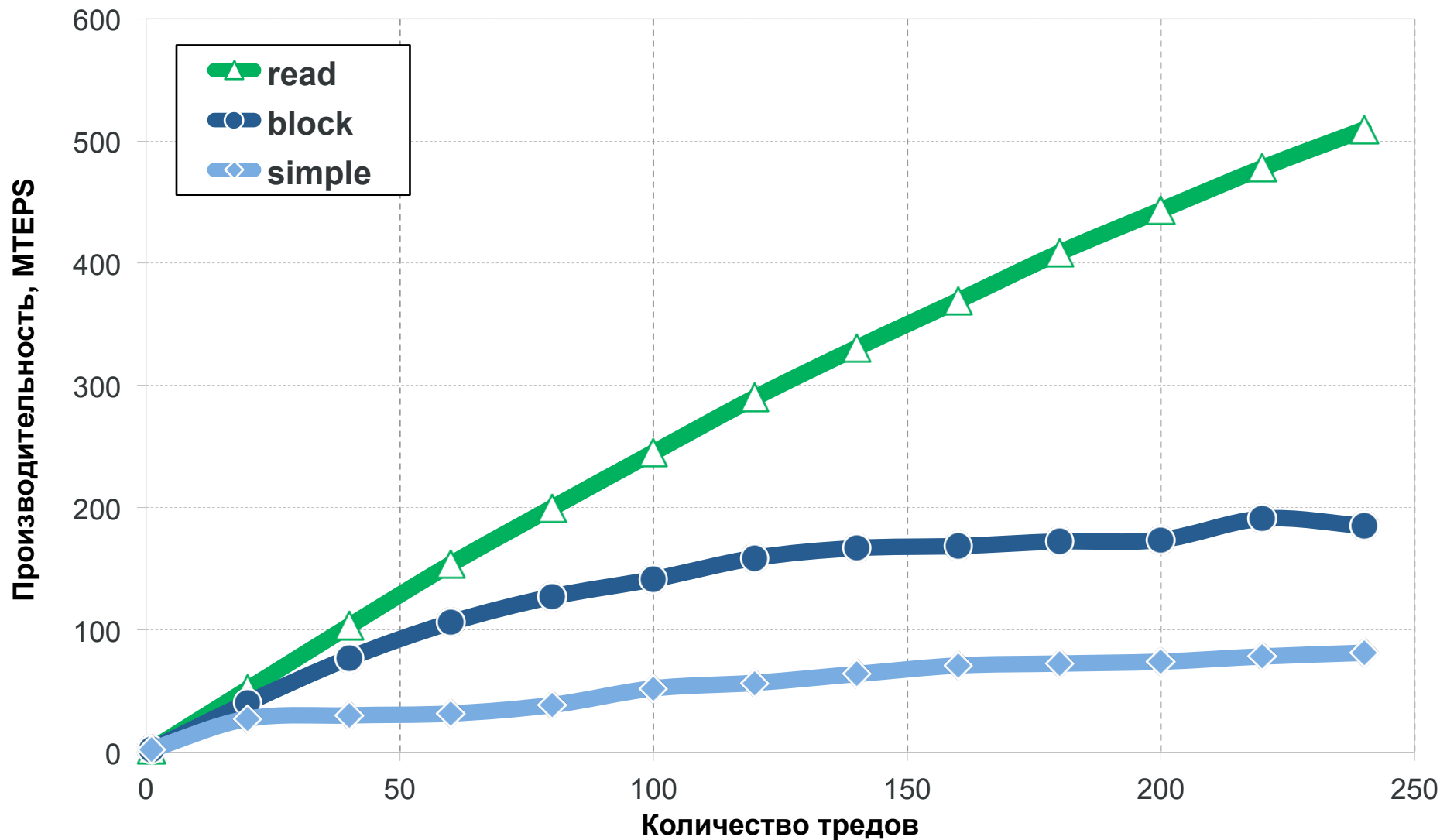


массив смежных вершин

```

#pragma omp parallel for reduction (...)
for all vertex  $\in V$  do
  if levels[vertex]  $\neq$  numLevel then
    continue
  for all w: (vertex, w)  $\in E$  do
    if levels[w] == -1 then
      levels[w] = numLevel + 1
      nLevelVerts = nLevelVerts + 1
    end if
  end for
end for
    
```

Производительность алгоритмов simple, block и read в зависимости от числа используемых тредов на сопроцессоре Phi-5110P

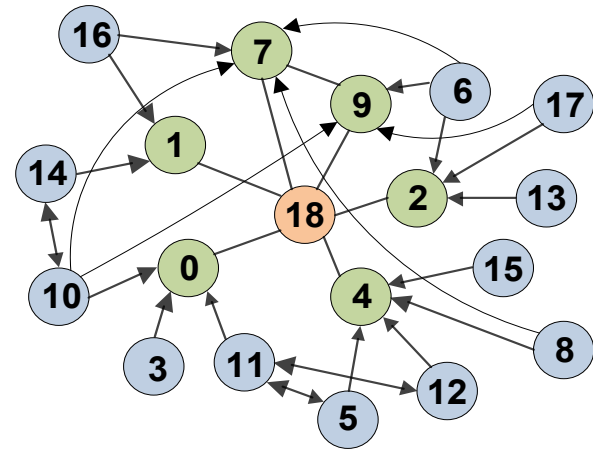


Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

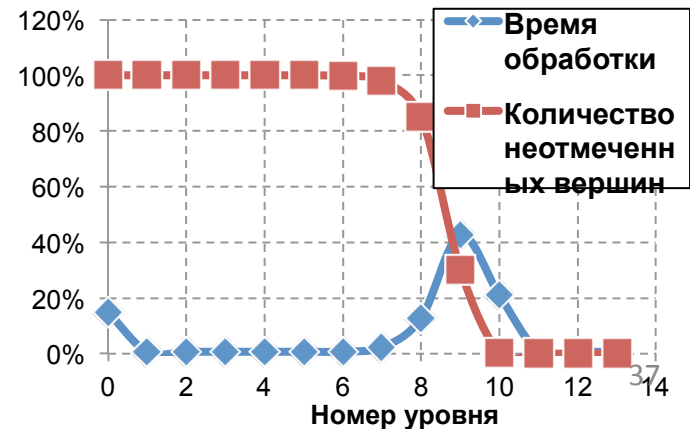
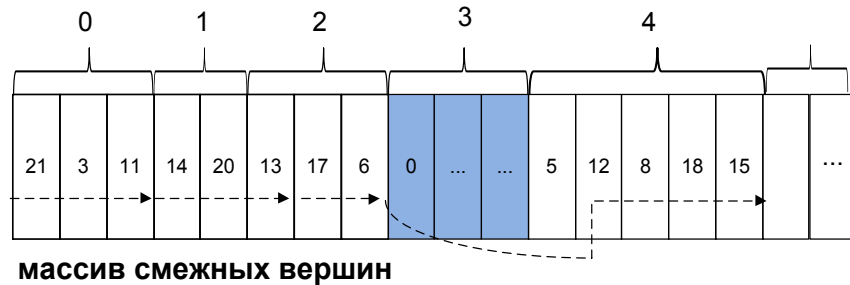
Алгоритм bottom-up-hybrid

```

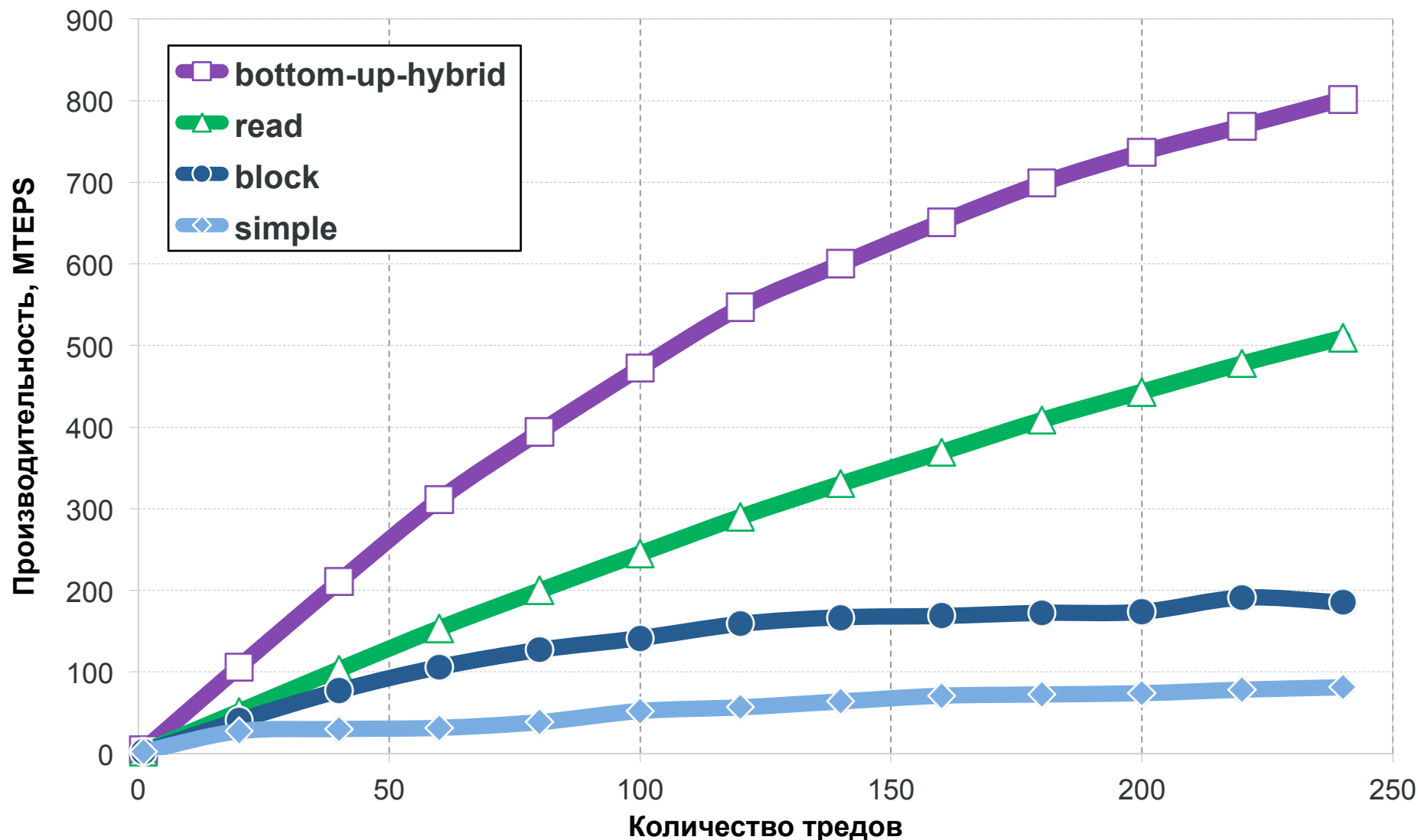
#pragma omp parallel for reduction (...)
for all vertex  $\in V$  do
    if levels[vertex] == -1 then
        for all w: (vertex, w)  $\in E$  do
            if levels[w] == numLevel then
                levels[vertex] = numLevel + 1
                nLevelVerts = nLevelVerts + 1
                break
            end if
        end for
    end if
end for
end for
    
```



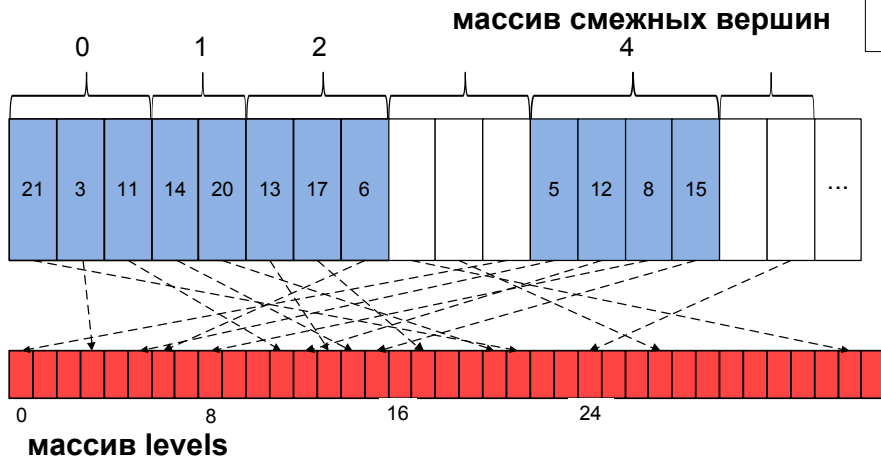
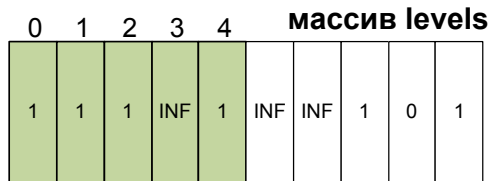
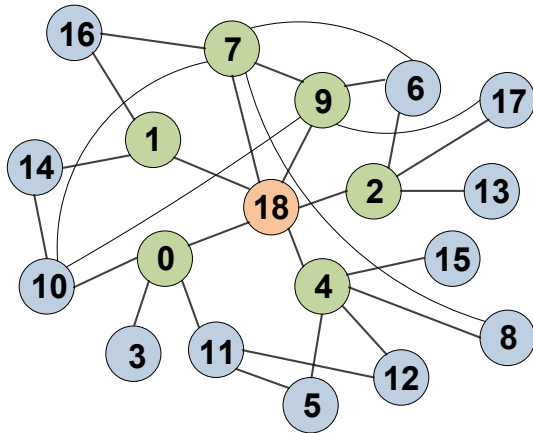
0	1	2	3	4	массив levels				
1	1	1	INF	1	INF	INF	1	0	1



Производительность алгоритмов simple, block, read и bottom-up-hybrid в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Недостатки алгоритмов read и bottom-up-hybrid



```
#pragma omp parallel for reduction  
(...)
```

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

```

if levels[vertex]  $\neq$  numLevel then
  continue

```

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

```

if levels[w] == -1 then
    levels[w] = numLevel + 1

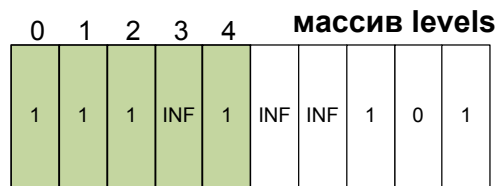
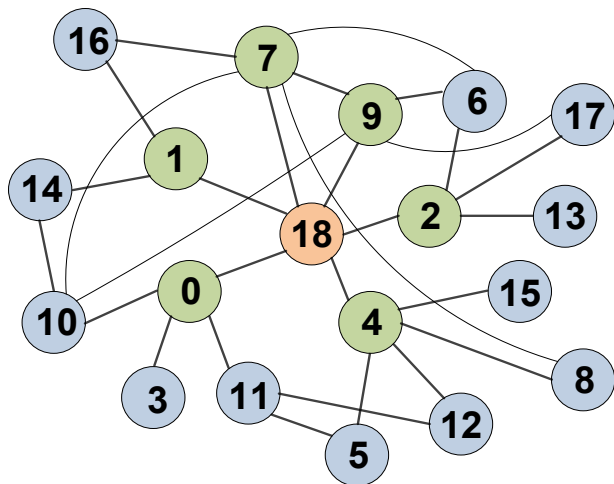
```

...

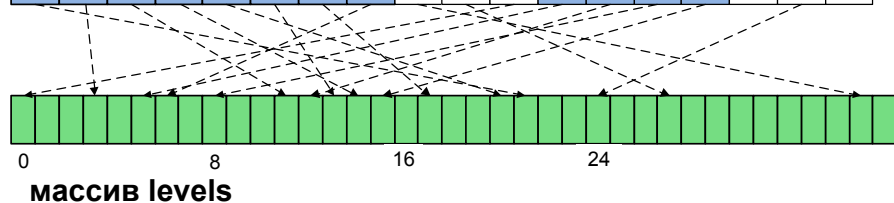
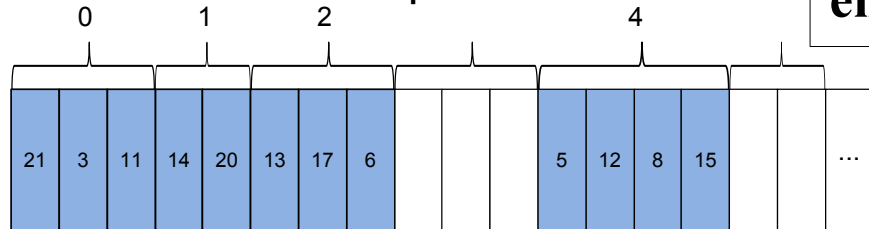
end if**end for****end for**

	SB	Phi-511 OP
Частота, ГГц	2.2	1.05
Задержка обращения в память (такты)	~150	~300

Решение: ручная развертка цикла + использование prefetch



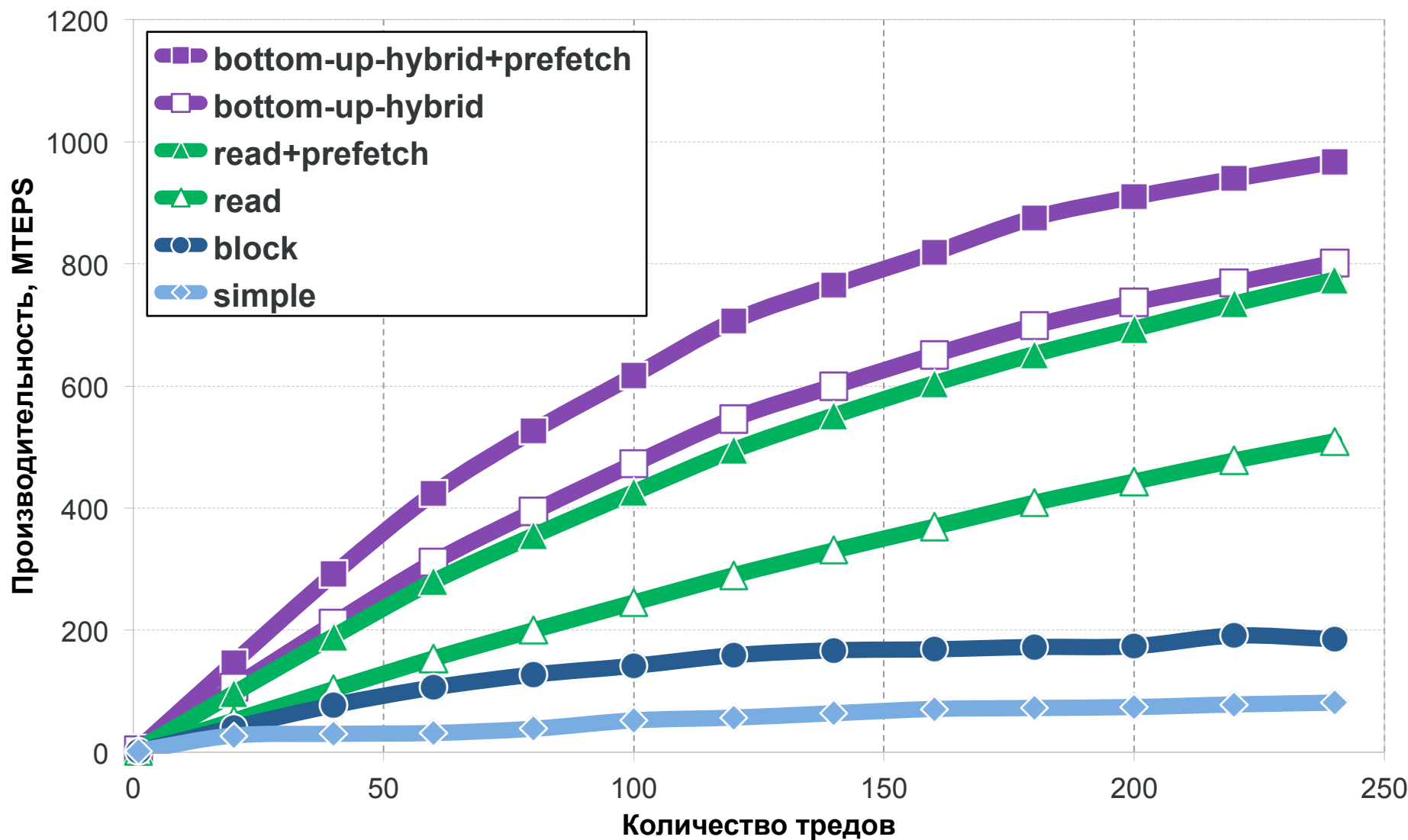
массив смежных вершин



```
#pragma omp parallel for reduction (...)
for all vertex ∈ V do
  if levels[vertex] ≠ numLevel then continue
  for all w: (vertex, w) ∈ E do
    prefetch(levels[w])
    ...
    if levels[w] == -1 then
      levels[w] = numLevel + 1
    ...
  end if
end for
end for
```

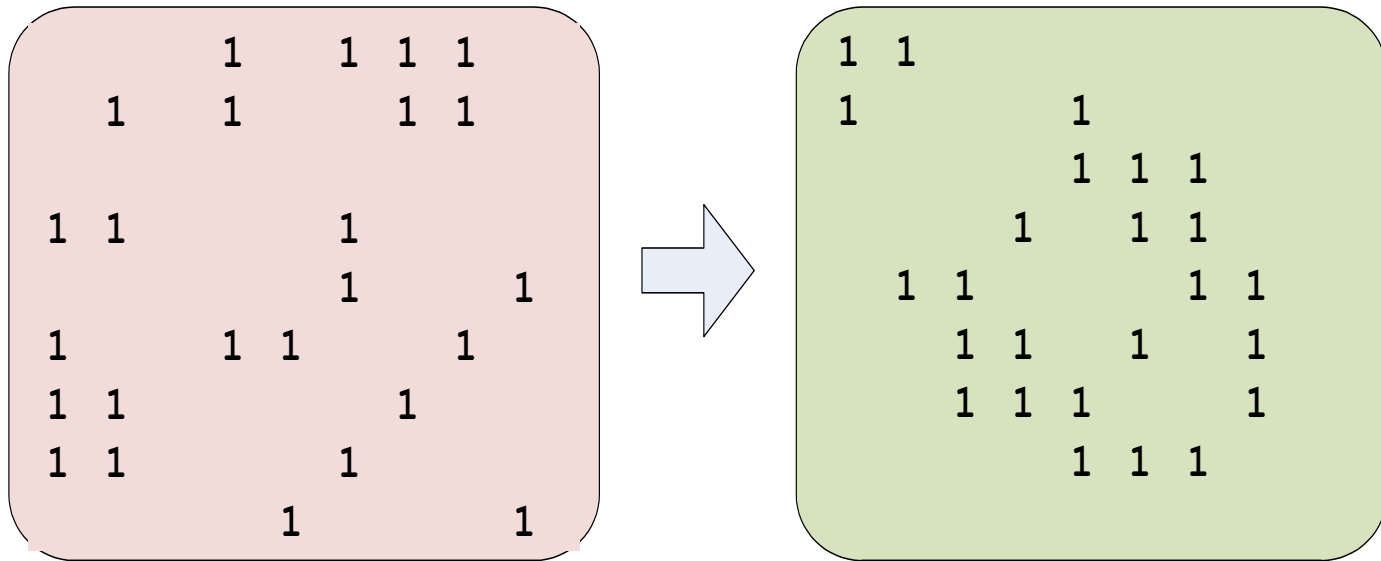
	SB	Phi-5110P
Пиковая ПС памяти, ГБ/с	51	352
ПС чтения из памяти, ГБ/с; Последовательный / случайный доступ	42 / 3.3	183 / 3.8
Задержка, тактов	200	300

Производительность алгоритмов simple, block, read и bottom-up-hybrid с префетчем в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



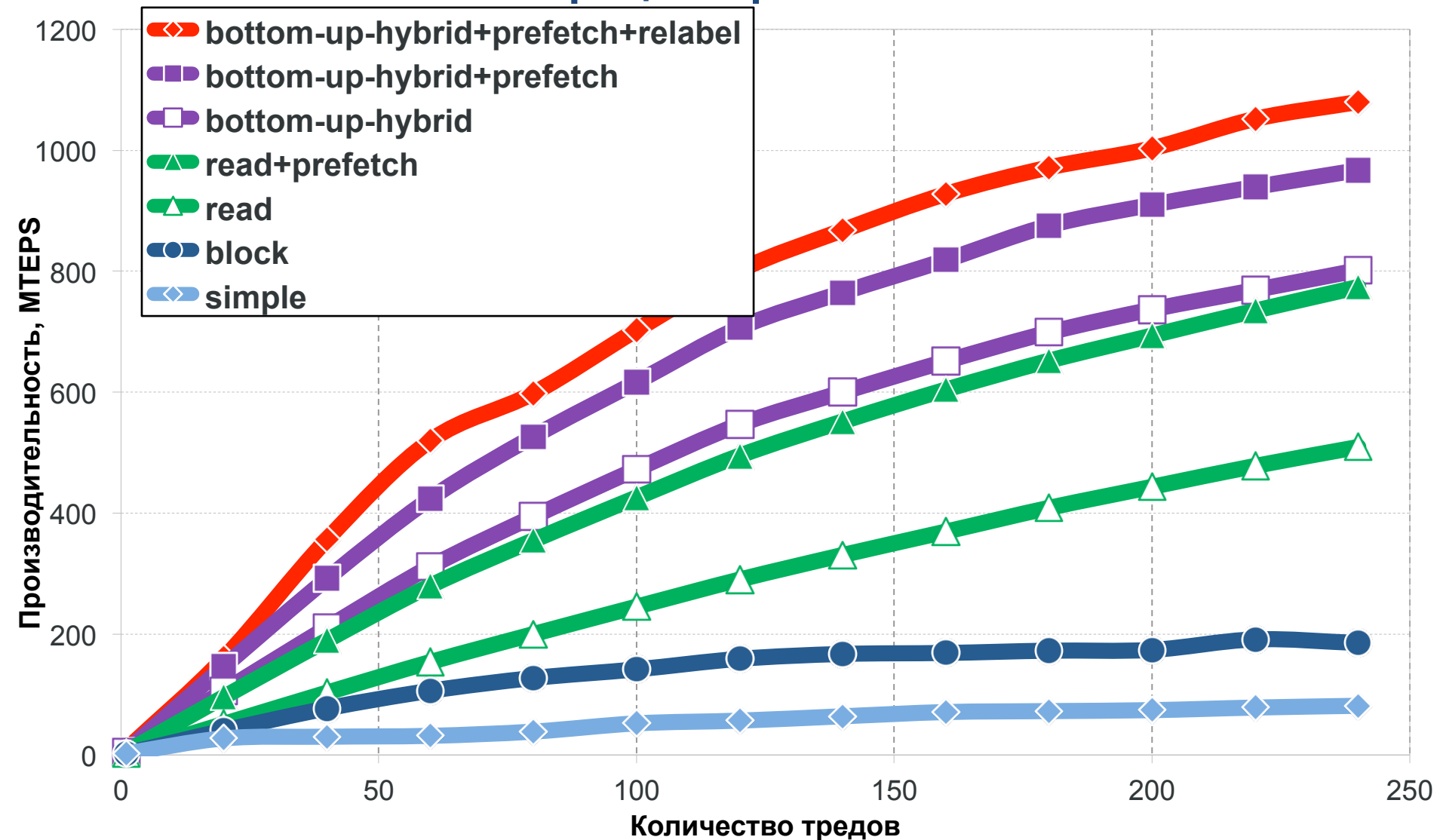
Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Улучшение локализации: перестановка вершин



- Матрица смежности приводится к ленточному виду с уменьшением ширины ленты (алгоритм Reverse Cuthill-McKee) => уменьшается количество кэш-промахов
- Списки смежных вершин сортируются => уменьшается количество промахов в TLB
- Использование больших страниц

Производительность различных алгоритмов, с префетчем и перестановками в зависимости от числа используемых тредов на сопроцессоре Phi-5110P



Число вершин в графе: $N = 2^{27}$ (134 млн), средняя связность вершины: $k = 8$

Распараллеливание: дисбаланс вычислительной нагрузки

- **Проблема: неравномерность** итераций циклов

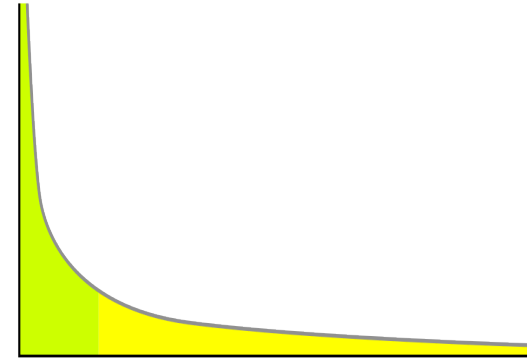
```
# pragma omp parallel for
```

```
for (int u = 0; u < G->n; u++)
```

```
    for (int j = G->rowsIndices[u]; j < rowsIndices[u+1]; j++) {
```

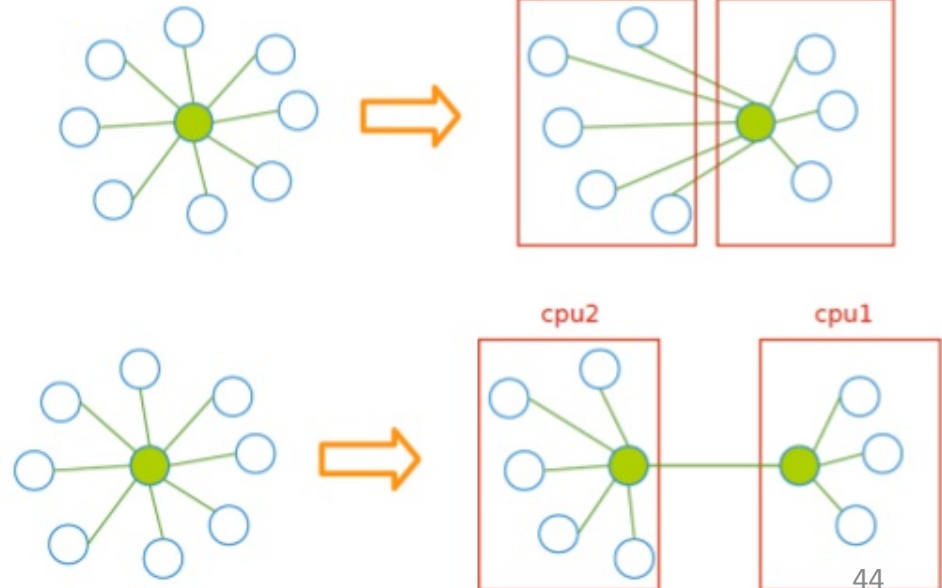
```
        .....
```

```
    }
```



- **Решение 1:** #pragma omp parallel for **schedule (guided)** – для **динамического** распределения вершин по тредам

- **Решение 2:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин



Большой объем памяти

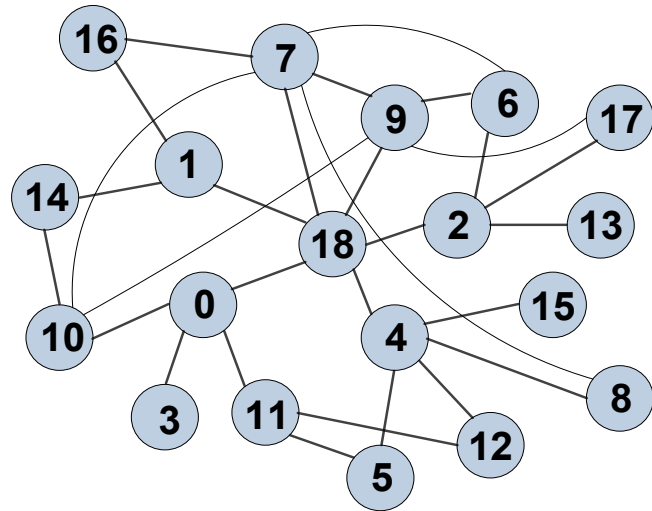
- **Проблема:** постоянная смена данных в кэше, низкие характеристики при случайном доступе
- **Решения** на этапе предобработки:
 - Хранение только половины графа (для неориентированного)
 - Удаление кратных ребер
 - Перестановка вершин (Cuthill-McKee)
 - Сжатие данных
 - `edge_id_t: uint64_t --> uint32_t`
 - Сортировка ребер каждой вершины
 - Сортировка всех ребер графа

Резюме: проблемы и подходы к решению задач в рамках одного узла

- Выбор оптимального представления графа
- По возможности организация последовательного доступа к данным
- По возможности избегать использовать межпоточковые синхронизации
- Стремиться работать не на задержке обращений к памяти, а на темпе
- Улучшение локализации
- Алгоритмические оптимизации
- Сжатие данных
- Аккуратная работа с памятью внутри NUMA-вычислительного узла
- Балансировка нагрузки
- Аккуратно измерять производительность

Проблемы и подходы к решению графовых задач на распределенной памяти

Представление графа

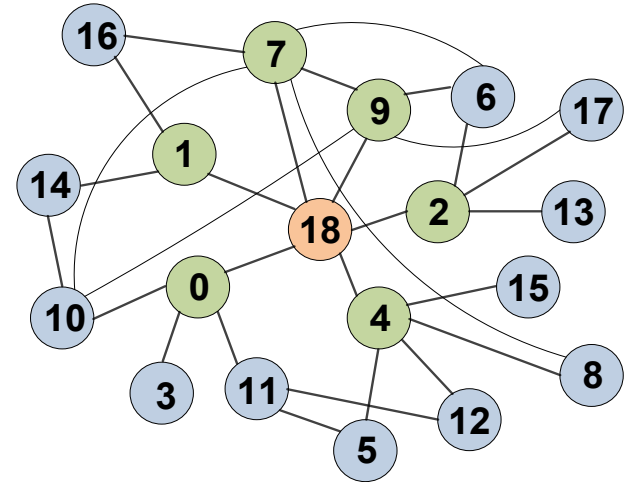


	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1				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

Поиск вширь в графе, распределенная версия

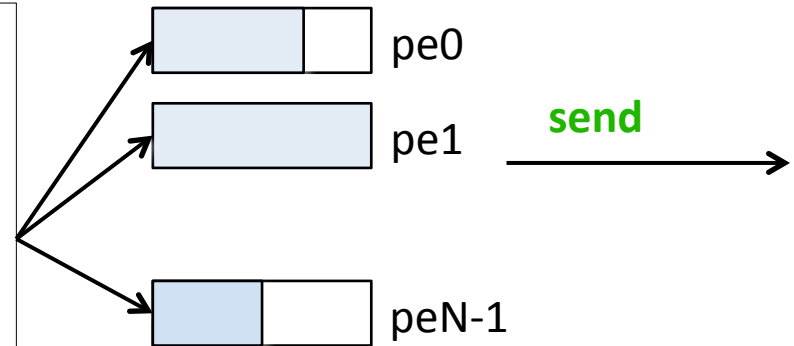
```
function ProcessQueue(Q, E)
  for all vertex  $\in$  Q do
    for all w : (vertex, w)  $\in$  E do
      send vertex, w to owner(w)
    end for
  end for
end function
```

```
function Receive(vertex, w)
  if w  $\notin$  Visited then
     $Q_{\text{next}} = Q_{\text{next}} \cup w$ 
    Visited = Visited  $\cup$  w
    Parents(w) = vertex
  end if
end function
```



Поиск вширь в графе, агрегация сообщений

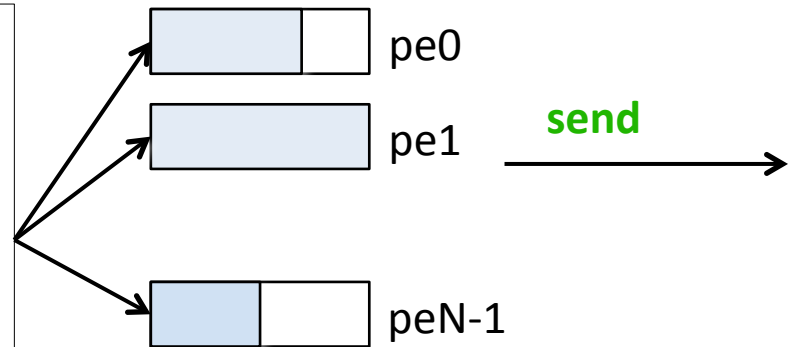
```
function ProcessQueue(Q, E)
  for all vertex  $\in$  Q do
    for all w : (vertex, w)  $\in$  E do
      send vertex, w to owner(w)
    end for
  end for
end function
```



```
function Receive(vertex, w)
  if w  $\notin$  Visited then
     $Q_{next} = Q_{next} \cup w$ 
    Visited = Visited  $\cup$  w
    Parents(w) = vertex
  end if
end function
```

Поиск вширь в графе, параллельная отправка и прием

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

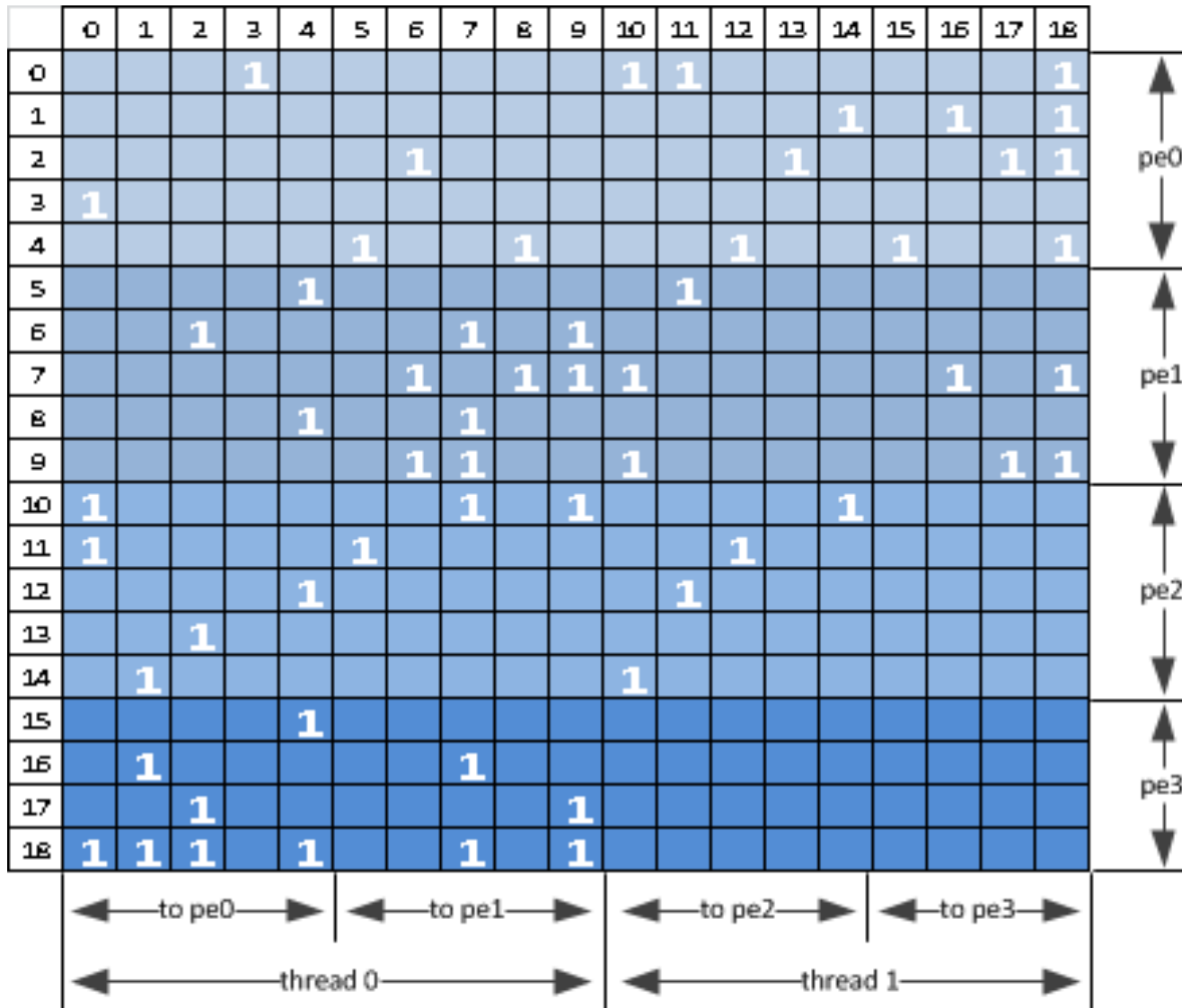


thread0

thread1

```
function Receive(vertex, w)
  if w  $\notin$  Visited then
     $Q_{\text{next}} = Q_{\text{next}} \cup w$ 
    Visited = Visited  $\cup$  w
    Parents(w) = vertex
  end if
end function
```

Организация параллелизма потоков



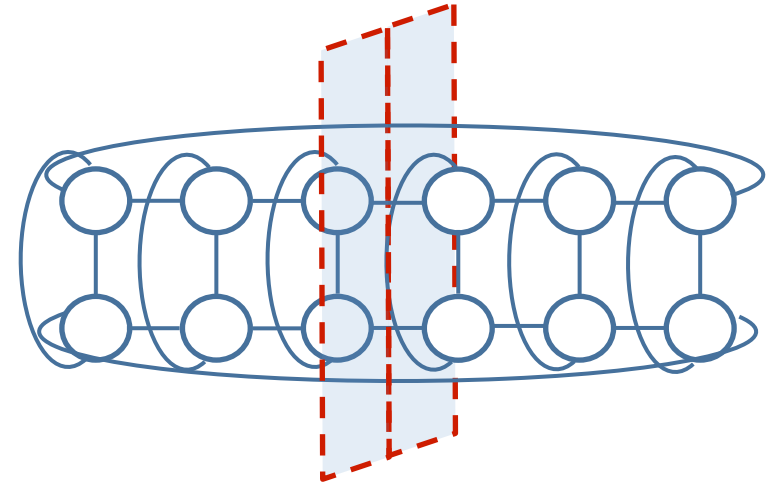
Хаотично расположенные вершины и ребра графа

	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										↑

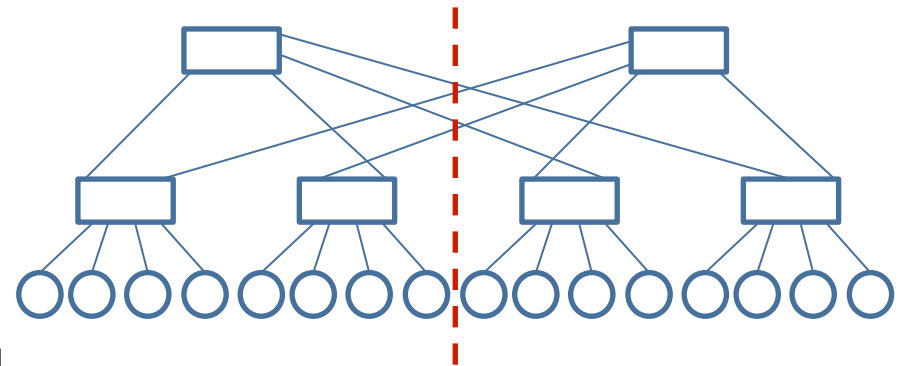
Шаблон обменов all-to-all

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 $\frac{1}{2}$
- Посылают все узлы – для линейной масштабируемости требуется $N/2$ линков в бисекционной плоскости



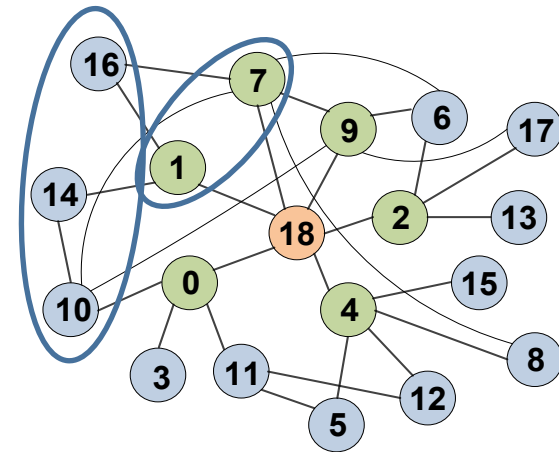
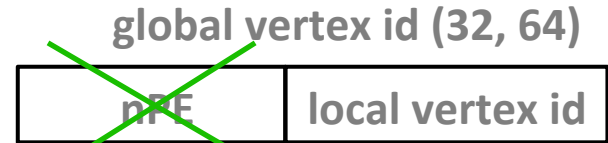
Бисекция тора = $2N/N_{\max}$



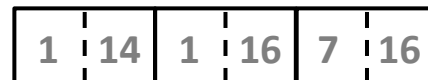
**Бисекция жирного дерева
(half bisection) = $N/4$**

Уменьшение количества пересылаемых данных

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



пересылаемое сообщение



Графы реального мира.

Степенной закон

- WWW, Социальные сети, Биоинформатика

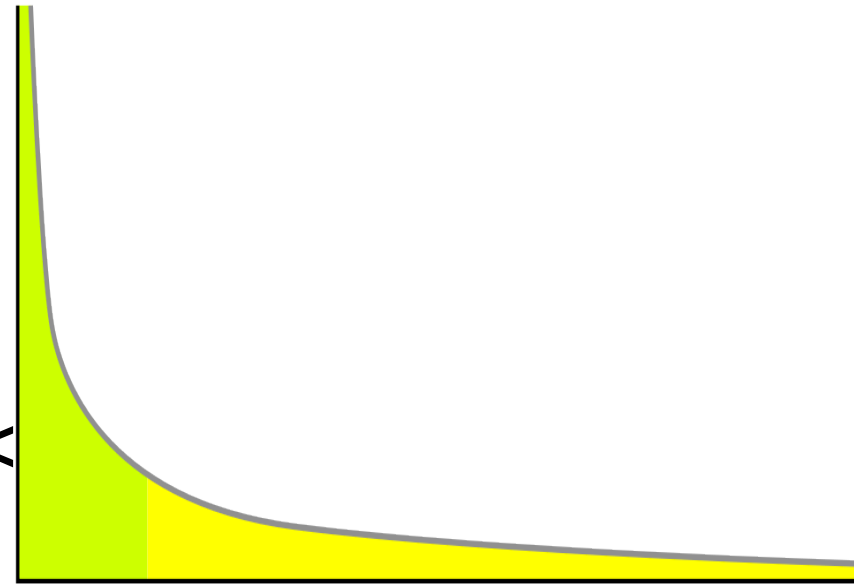
- Графы small-world

$L \sim \log N$,

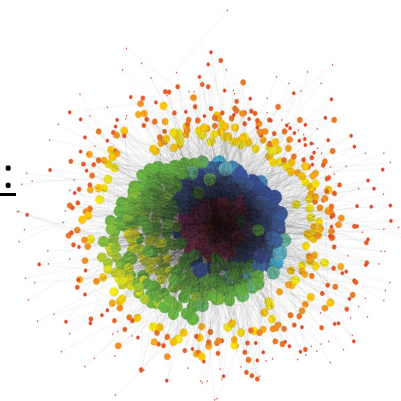
- scale-free – графы,
доля $P(k) \sim k^{-\tau}$, $2 < \tau < 3$

k – связность вершины

$L \sim \log \log N$



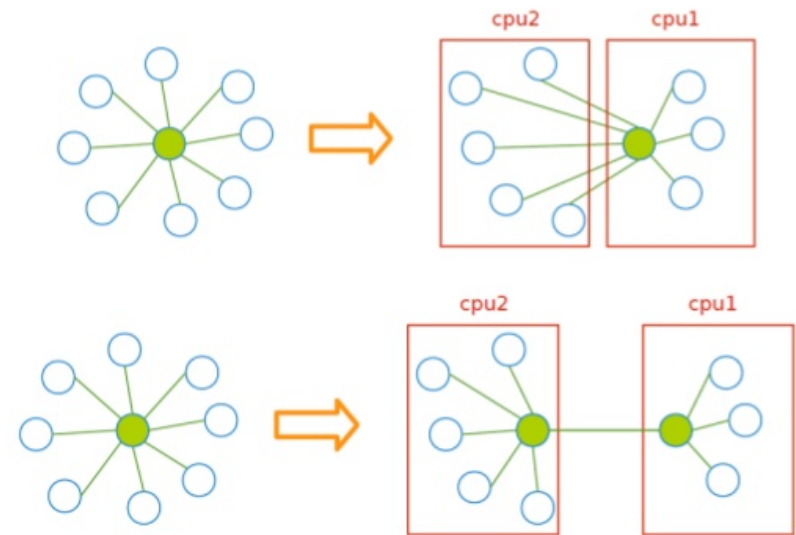
Граф Кронекера:



Балансировка нагрузки

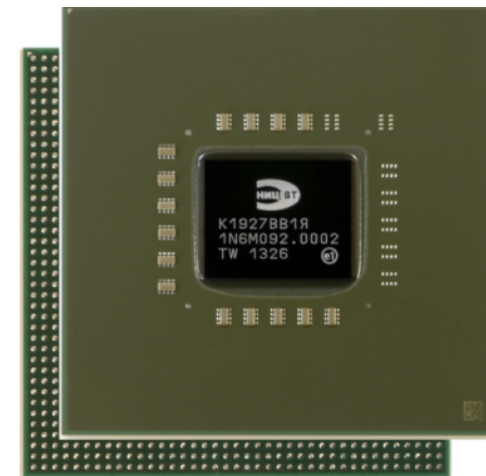
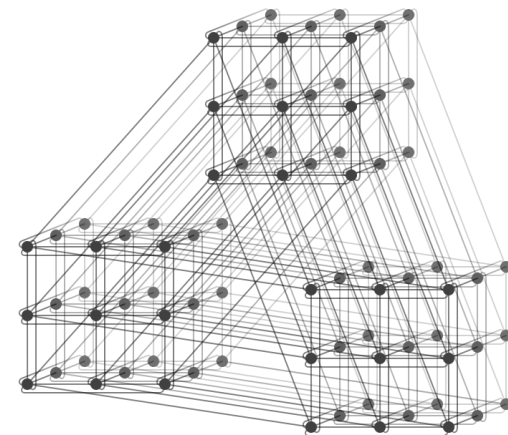
- При использовании большого числа вычислительных узлов особенно важна равномерная загрузка
- **Решение1:** На этапе предобработки выполнение процедуры Vertex-cut: разделение вершины и **разрезание** списков смежности вершин
- **Решение2:**

```
function ProcessQueue(Q, E)
  for all vertex  $\in Q$  do
    for all  $w : (vertex, w) \in E$  do
      if  $w \in \text{Heavy}$  then
         $\text{Out}^H = \text{Out}^H \cup w$ 
      else
        send vertex, w to owner(w)
      end if
    end for
  end for
  broadcast  $\text{Out}^H$ 
end function
```

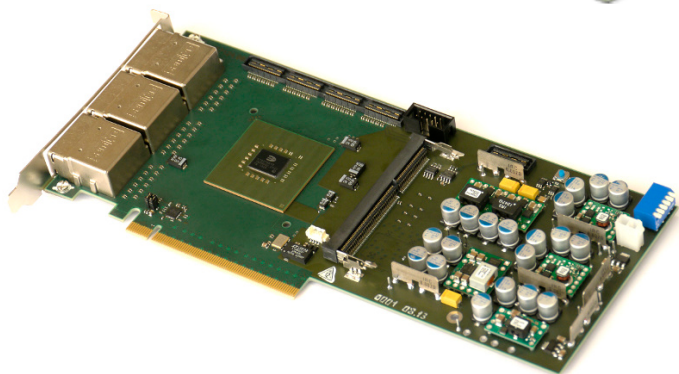
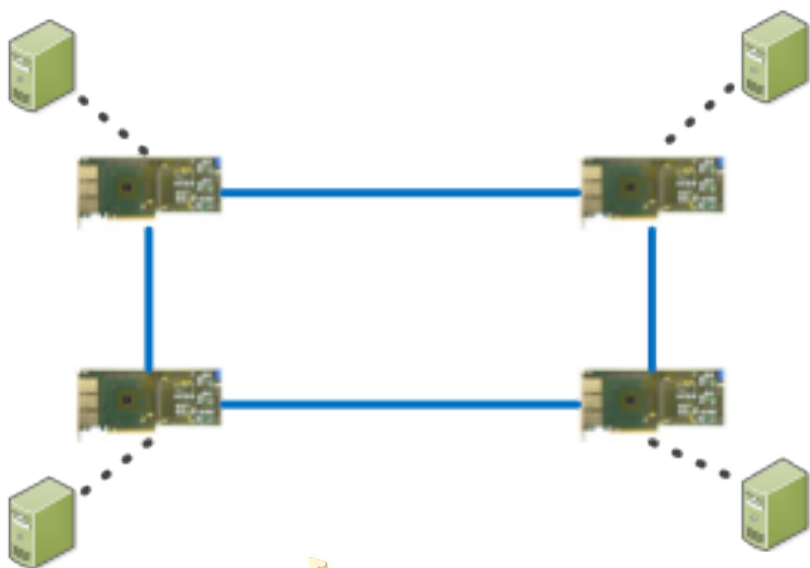


Ключевые особенности:

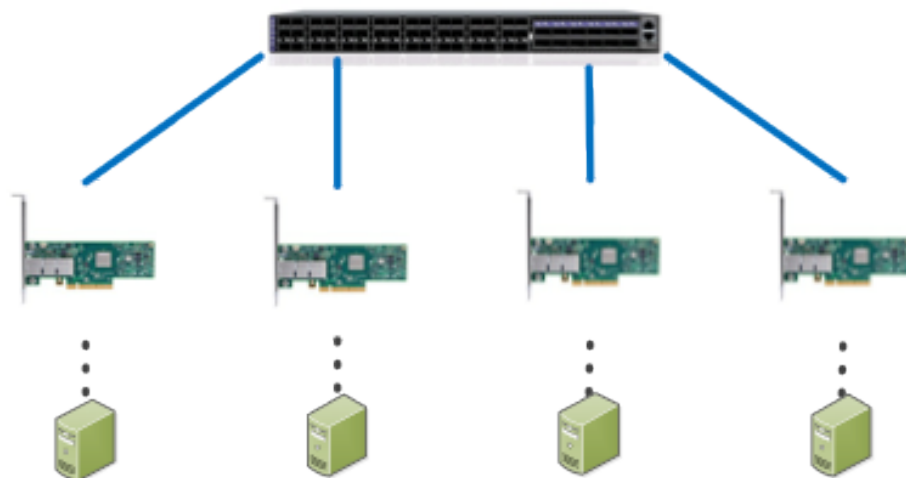
- Топология сети: 1D..4D-тор
- Адаптер на базе СБИС
- До 8 каналов связи с соседними узлами
- Прямой доступ в память удаленного узла (RDMA)
- Поддержка многоядерности
- Адаптивная передача пакетов
- Задержка на MPI ping-pong: 0,85 мкс
- Задержка на хоп: 130 нс
- Масштабирование: до 32K узлов
- Энергопотребление до 20 Вт
- Различные физические среды передачи данных



Ангара Тор

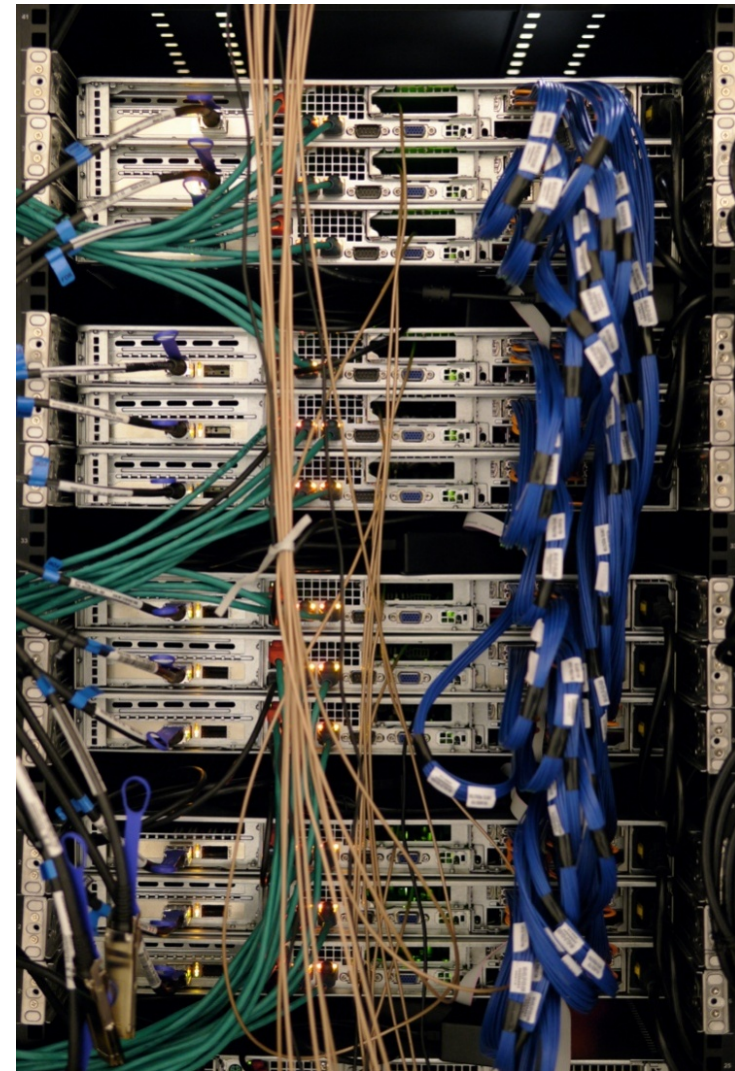


Infiniband Жирное дерево



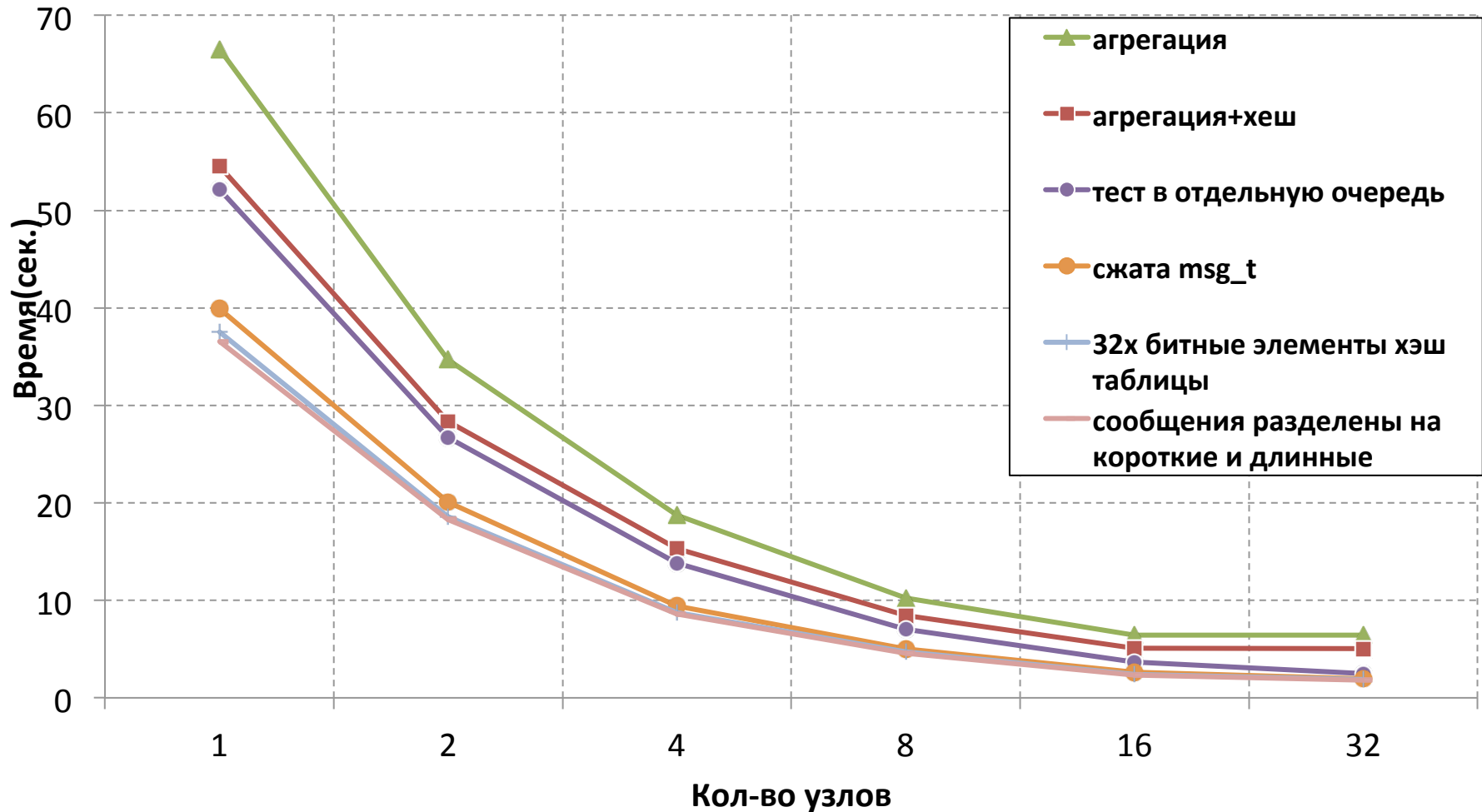
Сеть	Mellanox IB FDR 4x	Ангара	Mellanox IB EDR 4x	Intel OmniPath	Ангара-2
Год выпуска	2011	2013	2015	2015	2018
№ в Top500	13	—	20	6	
Топология	fat tree / kD-top	4D-top	fat tree / kD-top	fat tree	модиф. 4D-top
Задержка MPI, мкс	1	0.85	1	1	< 0.8
Задержка на хоп сети, нс	— / 250	129	н/д	н/д	< 100

- **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

Вопросы?

alxdr.semenov@gmail.com