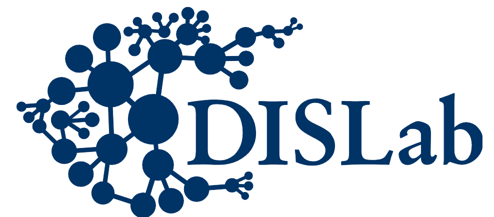
Параллельная обработка больших графов

Занятие 10

А.С. Семенов

dislab.org

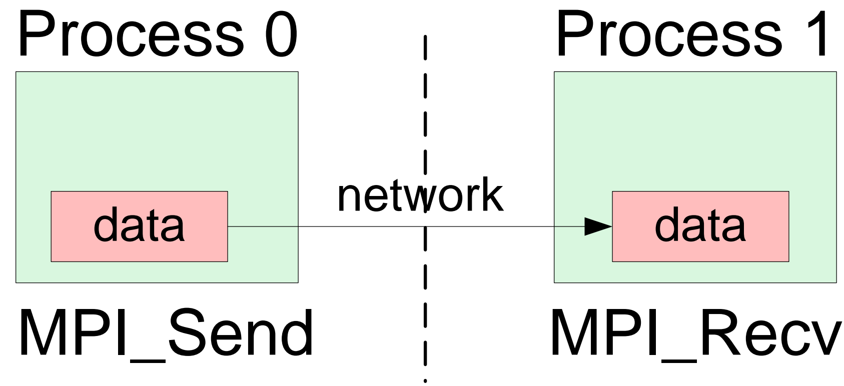


Message Passing Interface, MPI

- Основная цель: обеспечение переносимости программ, эффективная реализация, поддержка гетерогенных параллельных архитектур
- MPI-1 draft 1992, MPI-1.1 – 1995
- MPI-2 draft 1996, MPI-2.2 – 2009
- MPI-3 draft 2010
- Поддерживается на всех суперкомпьютерах, C, Fortran
- MPI-1 – около 125 функций, но часто используется лишь небольшое подмножество
  - Существуют эффективные реализации, используется большинством приложений
- Состав MPI-1:
  - Функции инициализации и завершения
  - Операции точка-точка
  - Коллективные операции

Операции точка-точка

- ***MPI_Send (sendbuf, count, type, dest, tag, comm, ierr)*** – блокирующая посылка, возвращает значение, когда данные посланы из буфера
- ***MPI_Recv (recvbuf, count, type, source, tag, comm, status, ierr)*** – блокирующий прием, возвращает значение, когда данные получены в буфер



Тип MPI	Тип Fortran
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER(1)

Протоколы передачи данных в MPI: Eager

Сообщение сразу посылается на узел-получатель, но, возможно, там буферизуется

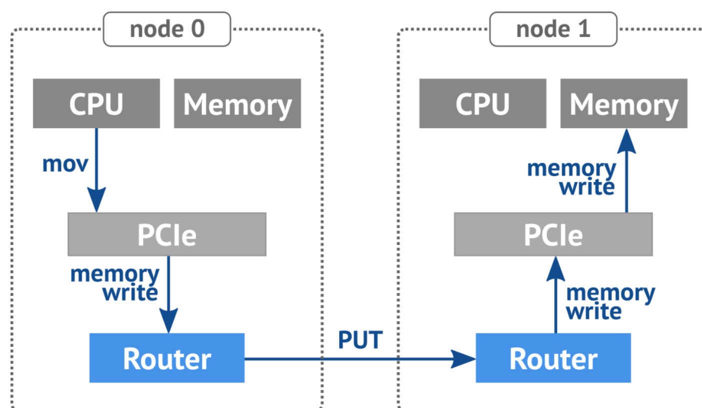
Преимущества:

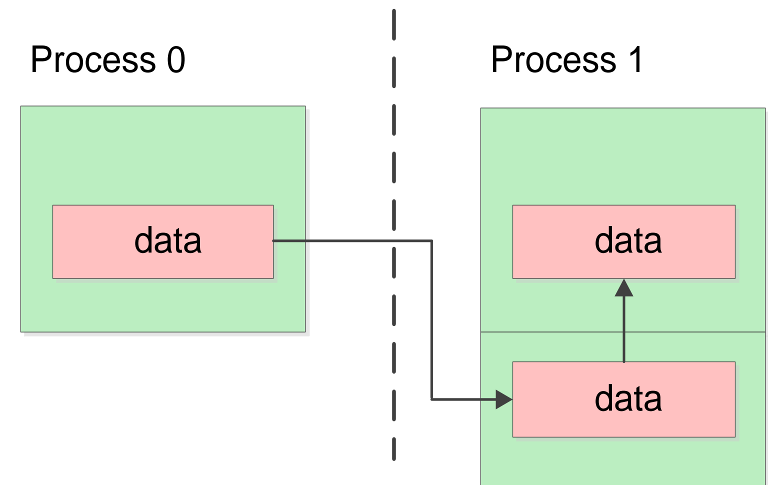
- Уменьшается коммуникационная задержка

Недостатки:

- Требуется буферизация – протокол не масштабируем
- Расход памяти, даже если это не требуется
- Возможно переполнение буфера
- Может потребоваться дополнительное копирование

Используется для коротких сообщений





Протоколы передачи данных в MPI: Rendezvous

Когда нельзя предсказать состояние буфера приема или когда достигнуты ограничения eager

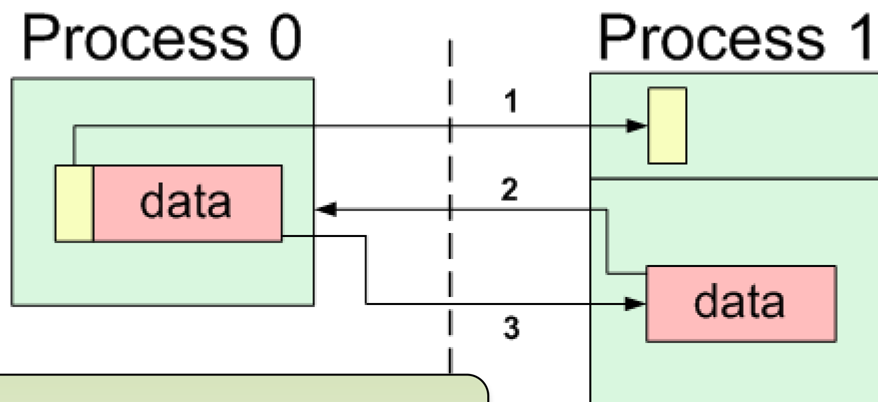
Преимущества:

- Масштабируем по сравнению с eager
- Требуется небольшой объем памяти для хранения метаданных сообщения
- Устойчивость к переполнению памяти на принимающем узле
- Отсутствие копирования данных

Недостатки:

- Растет задержка передачи сообщения из-за дополнительных коммуникаций

Используется для длинных сообщений



Переменные окружения:

- EAGER_LIMIT
- BUFFER_MEM

Эффективность программ:

- Выдавать Recv раньше Send

Буферизация сообщений

Системная буферизация:

- На стороне отправителя
- На стороне получателя
- Может отсутствовать
- При некоторых условиях может присутствовать, при некоторых – нет, например, eager и rendezvous

Преимущество системной буферизации:

- Повышение производительности за счет возможной асинхронной передачи
- Если операция приема не была выдана, отправитель может продолжить работу

Недостатки:

- Требуется дополнительное копирование
- Буфер – конечный ресурс. Переполнение буфера – аварийное завершение или простой
- Не очевидно для программиста, когда используется буфер – зависимость от реализации MPI
- Программа, работающая при одних условиях, может не работать при других

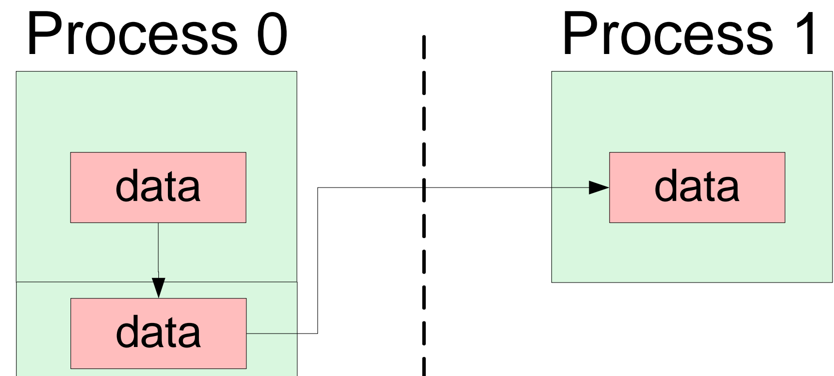
- Размер системного буфера может задаваться переменной окружения
- Правильная MPI-программа не полагается на размер буфера.
- Небезопасная программа в большинстве ситуаций может работать корректно

Пользовательская буферизация сообщений

- ***MPI_Bsend*** *(sendbuf, count, type, dest, tag, comm, ierr)*
- завершается, когда сообщение скопируется в буфер

Преимущества:
- Немедленное завершение (предсказуемость)
- Асинхронная передача

Недостатки:
- Требуется дополнительное копирование
- Требуется явно выделять буфер ***MPI_Buffer_attach*** *(buffer, size, ierr)* и контролировать его использование
- ***MPI_Buffer_detach*** *(buffer, size, ierr)*



Эффективность программ:
Из-за копирования лучше не использовать MPI_Bsend для длинных сообщений

Неблокирующие операции

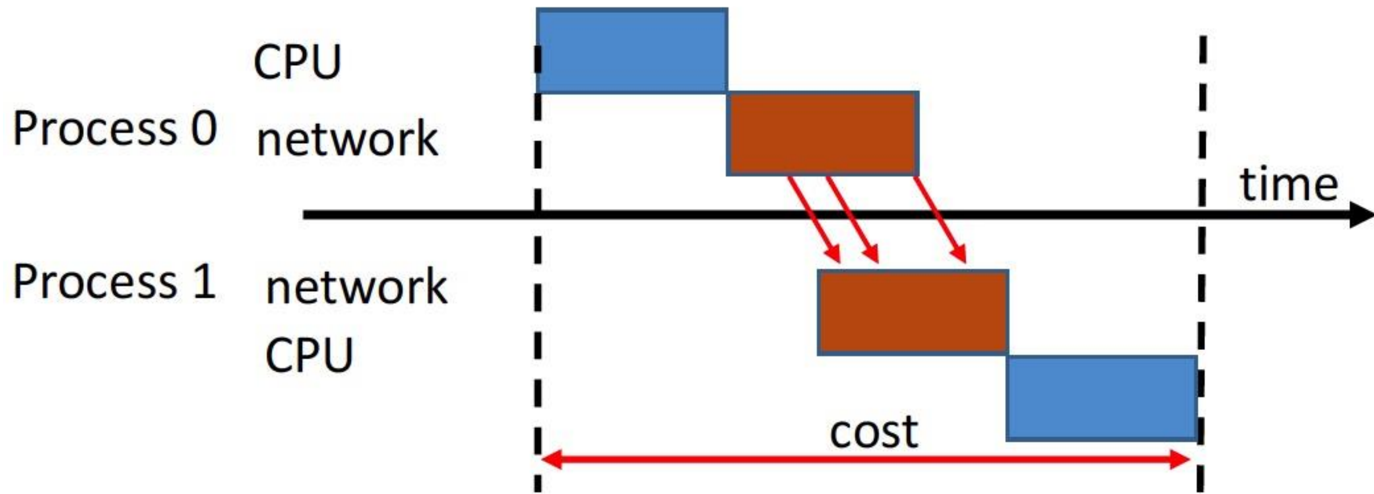
- ***MPI_Isend (sendbuf, count, type, dest, tag, comm, request)***
- ***MPI_Irecv (recvbuf, count, type, source, tag, comm, request)***
- Сразу возвращают управление
- Нельзя использовать буфер до тех пор, пока операция не закончится:
  - ***MPI_Wait (request, status)***
  - ***MPI_Waitany, MPI_Waitsome, MPI_Waitall***
- Функции тестирования ***MPI_Test, MPI_Testany, MPI_Testall, MPI_Testsome***
- Отсутствует лишняя буферизация
- Возможно выполнение коммуникаций на фоне вычислений и, как следствие, выигрыш в производительности

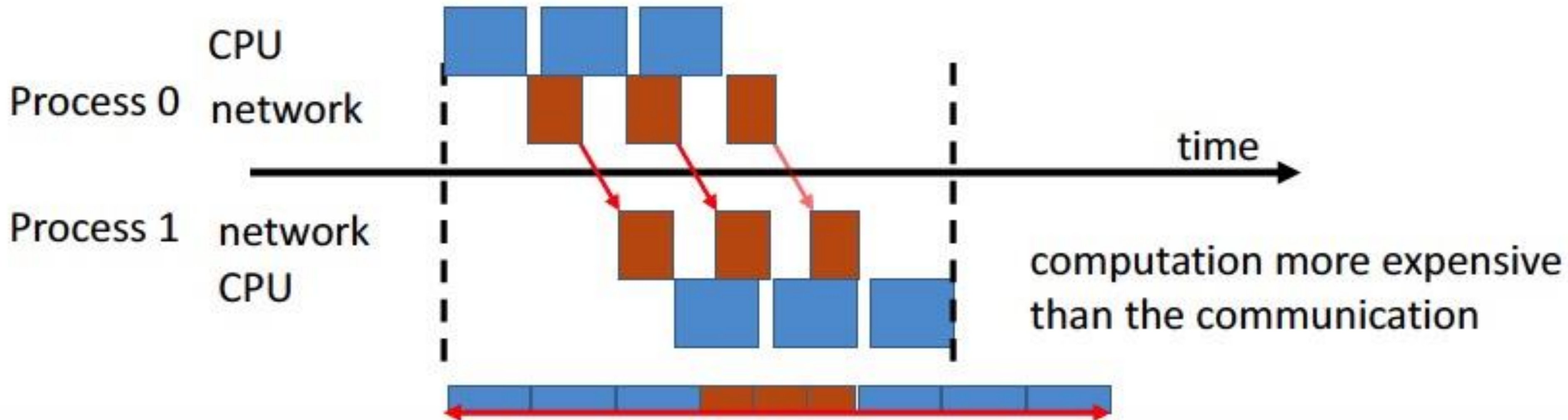
```
MPI_Irecv  ( A, count, MPI_DOUBLE,  src, 0, MPI_COMM_WORLD, reqs[0] );
MPI_Isend ( B, count, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD, reqs[1] );
/* do computation */
MPI_Waitall (2, reqs, stats );
```

Эффективность программ:

- Стремиться использовать неблокирующие операции

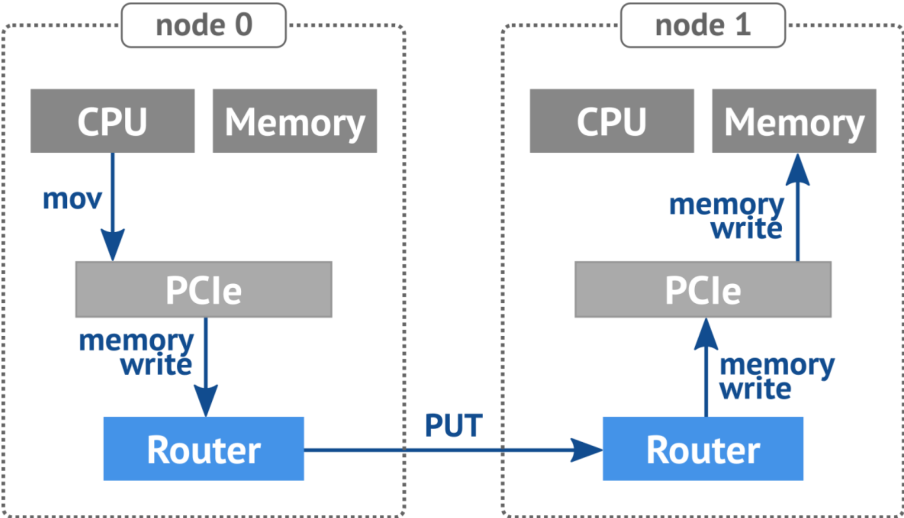
Блокирующие и неблокирующие операции



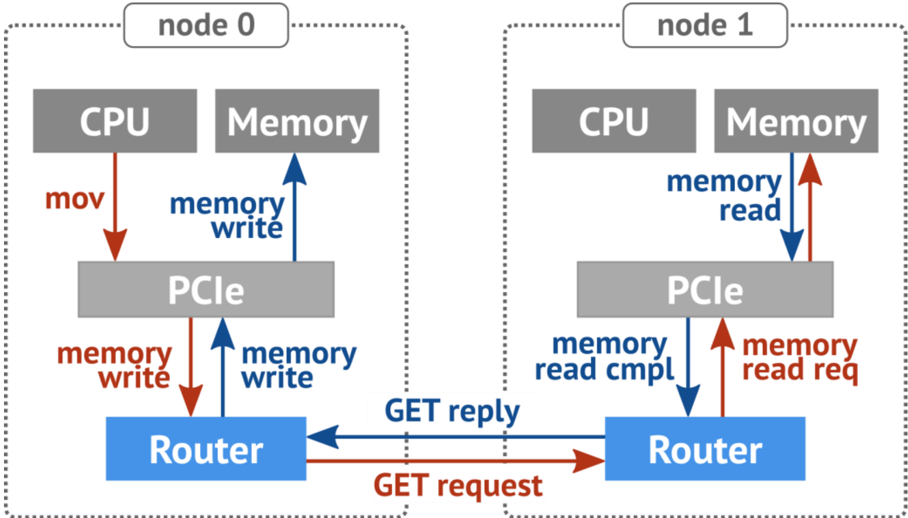


Механизм передачи RDMA. Сеть Ангара

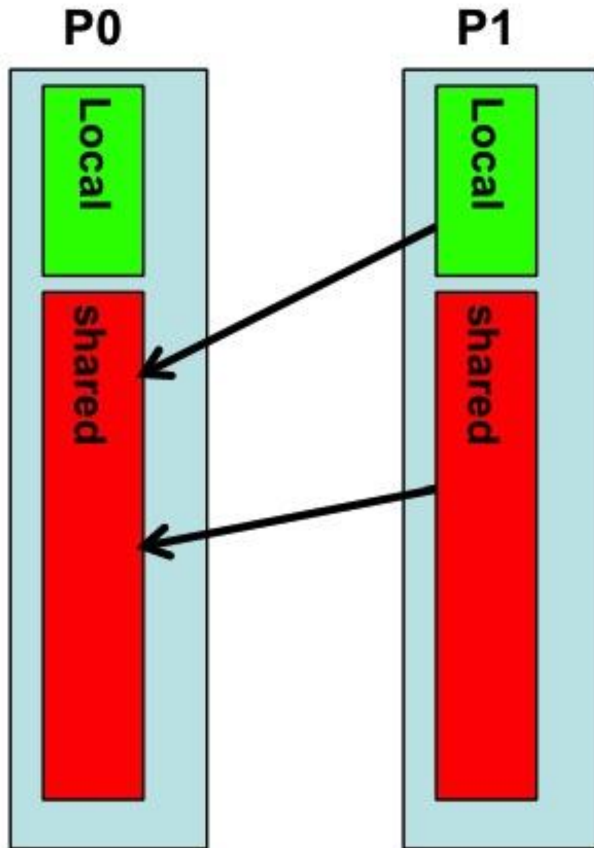
Удалённая запись

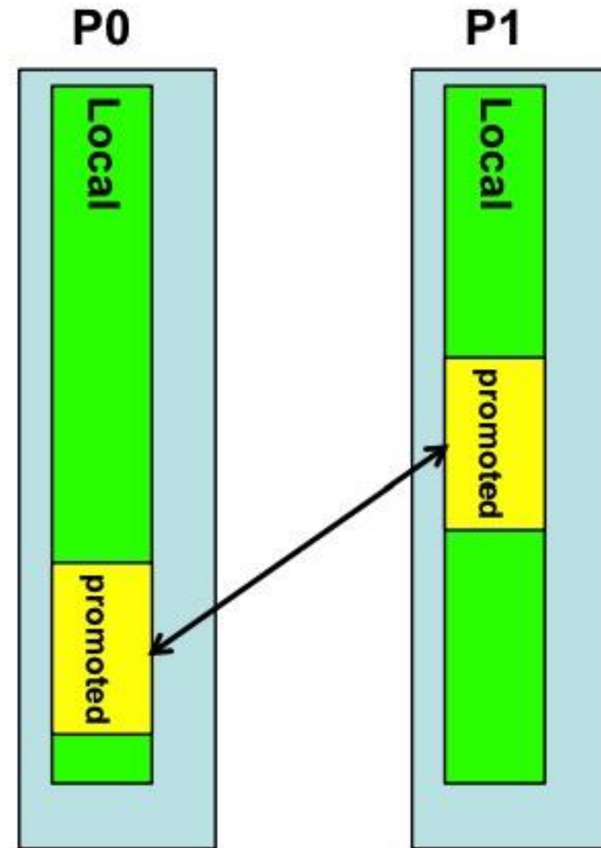


Удалённое чтение



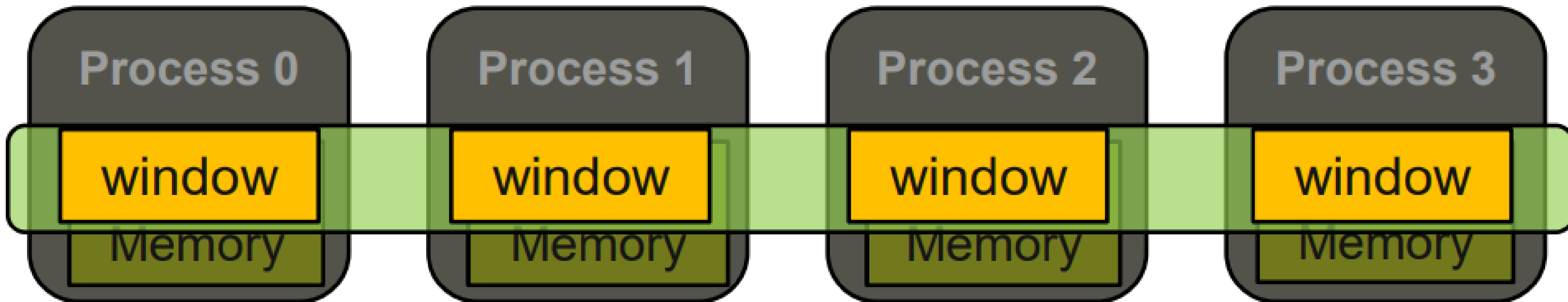
Односторонние и двусторонние коммуникации





MPI-3

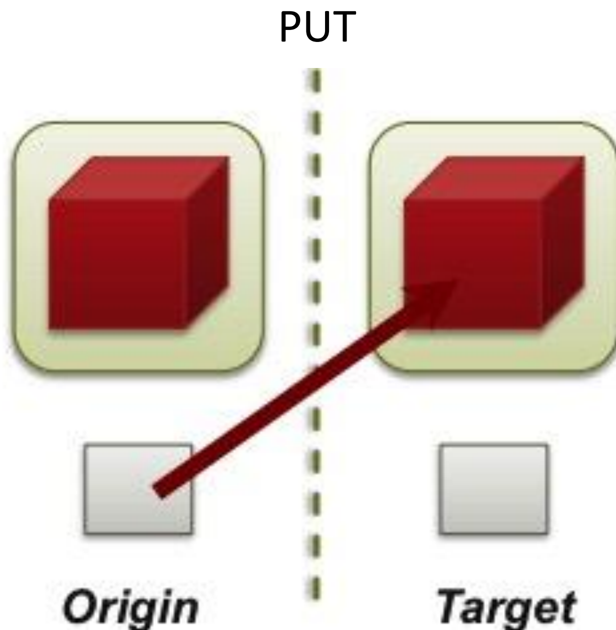
- ***MPI_Win_allocate (size, disp_unit, info, comm, base_ptr, win)*** – выделяет память и создает окно
- ***MPI_Win_create (base_ptr, size, disp_unit, info, comm, win)*** – создает окно для уже выделенной памяти

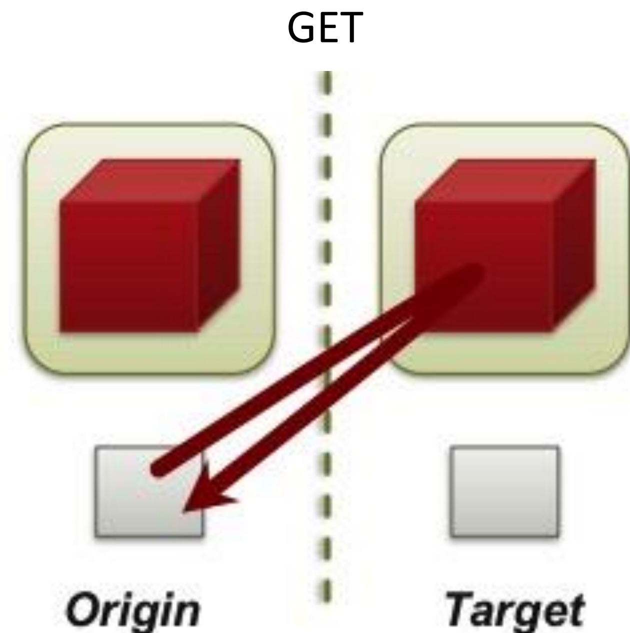


https://htor.inf.ethz.ch/publications/img/MPI_RMA_and_advanced_MPI.pdf

MPI-3

- ***MPI_Put (sendbuf, count, type, target, target_disp, target_count, target_type, win)***
- ***MPI_Rput (sendbuf, count, type, target, target_disp, target_count, target_type, win, request)***
  - *MPI_Test, MPI_Wait*
- ***MPI_Get (origin, origin_count, origin_dtype, target, target_count, target_dtype, win)***



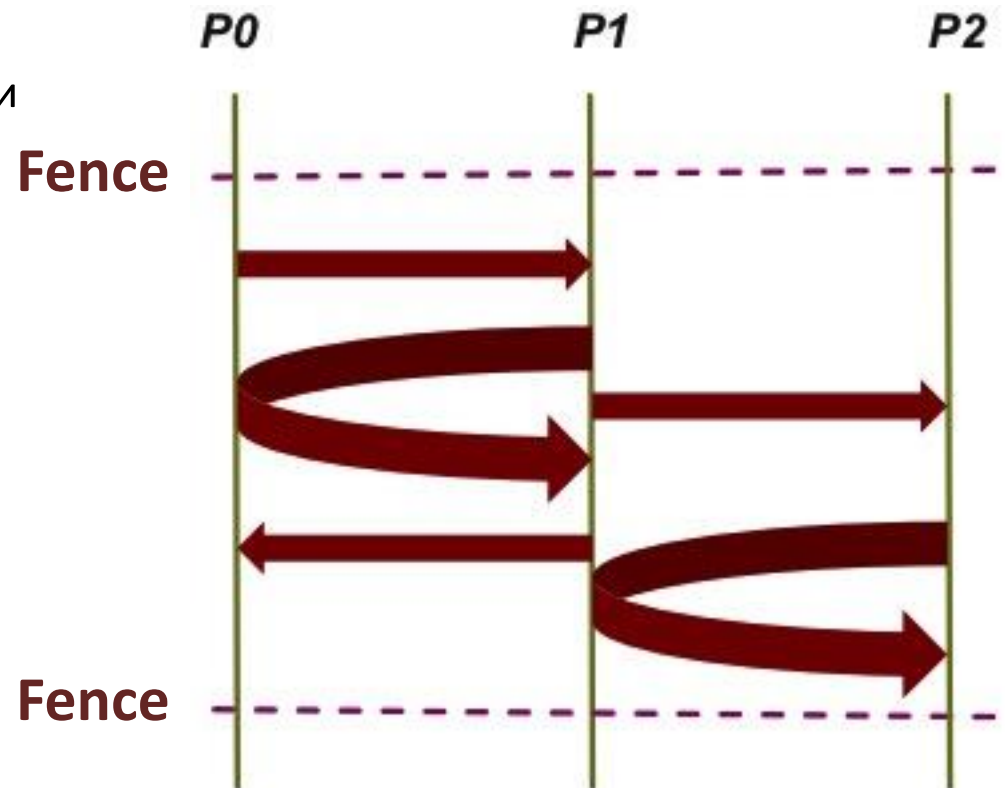


https://htor.inf.ethz.ch/publications/img/MPI_RMA_and_advanced_MPI.pdf

Коллективная синхронизация в MPI-3

- ***MPI_Win_fence (assert, win)***

Модель коллективной синхронизации



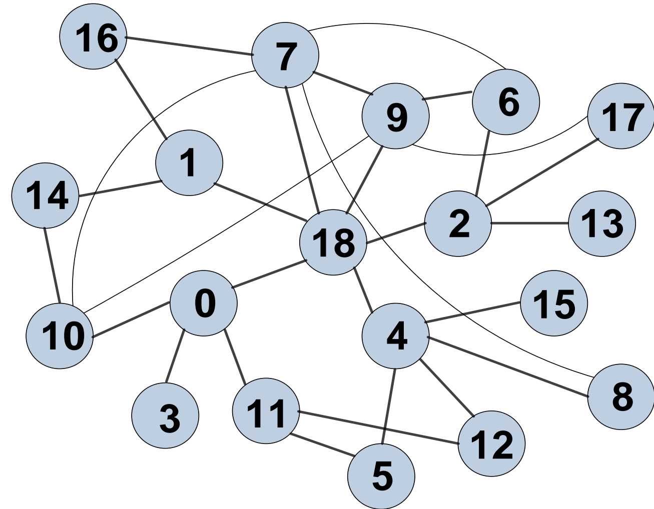
https://htor.inf.ethz.ch/publications/img/MPI_RMA_and_advanced_MPI.pdf

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

	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									

pe0 (rows 0–4), pe1 (rows 5–9), pe2 (rows 10–14), pe3 (rows 15–18)

1D, с чередованием

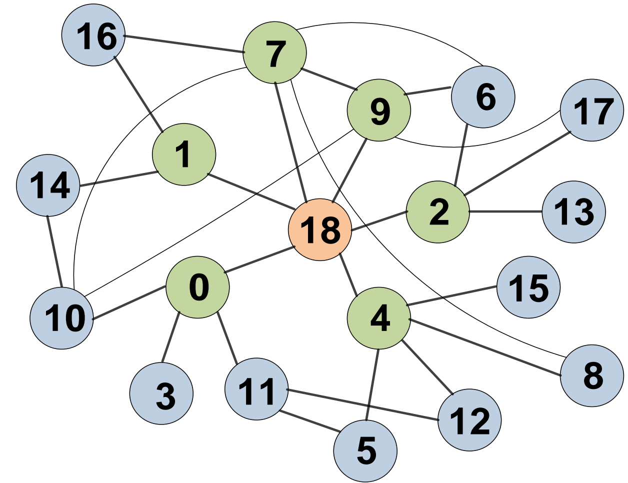
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1				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

2D

	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

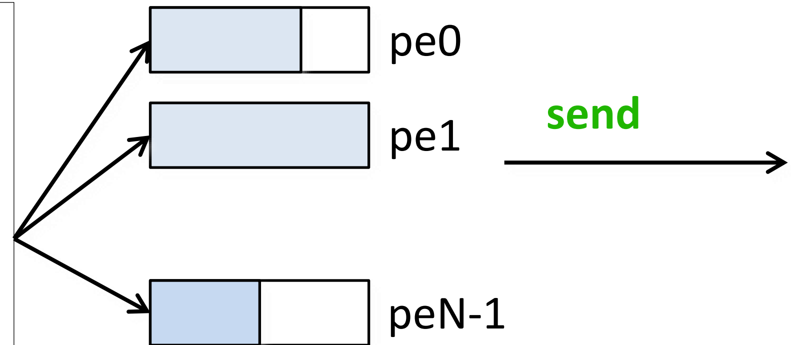
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
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

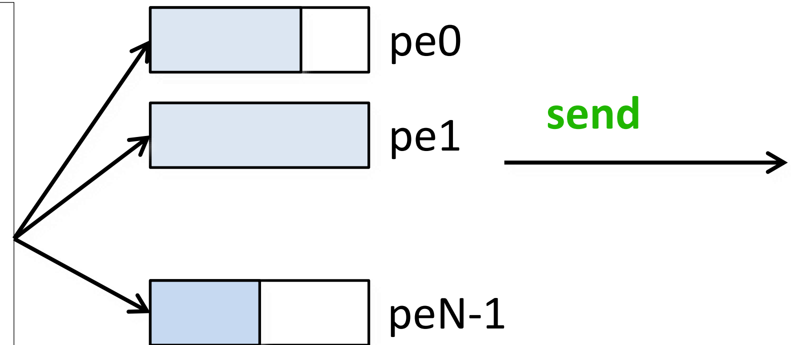
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```




```
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

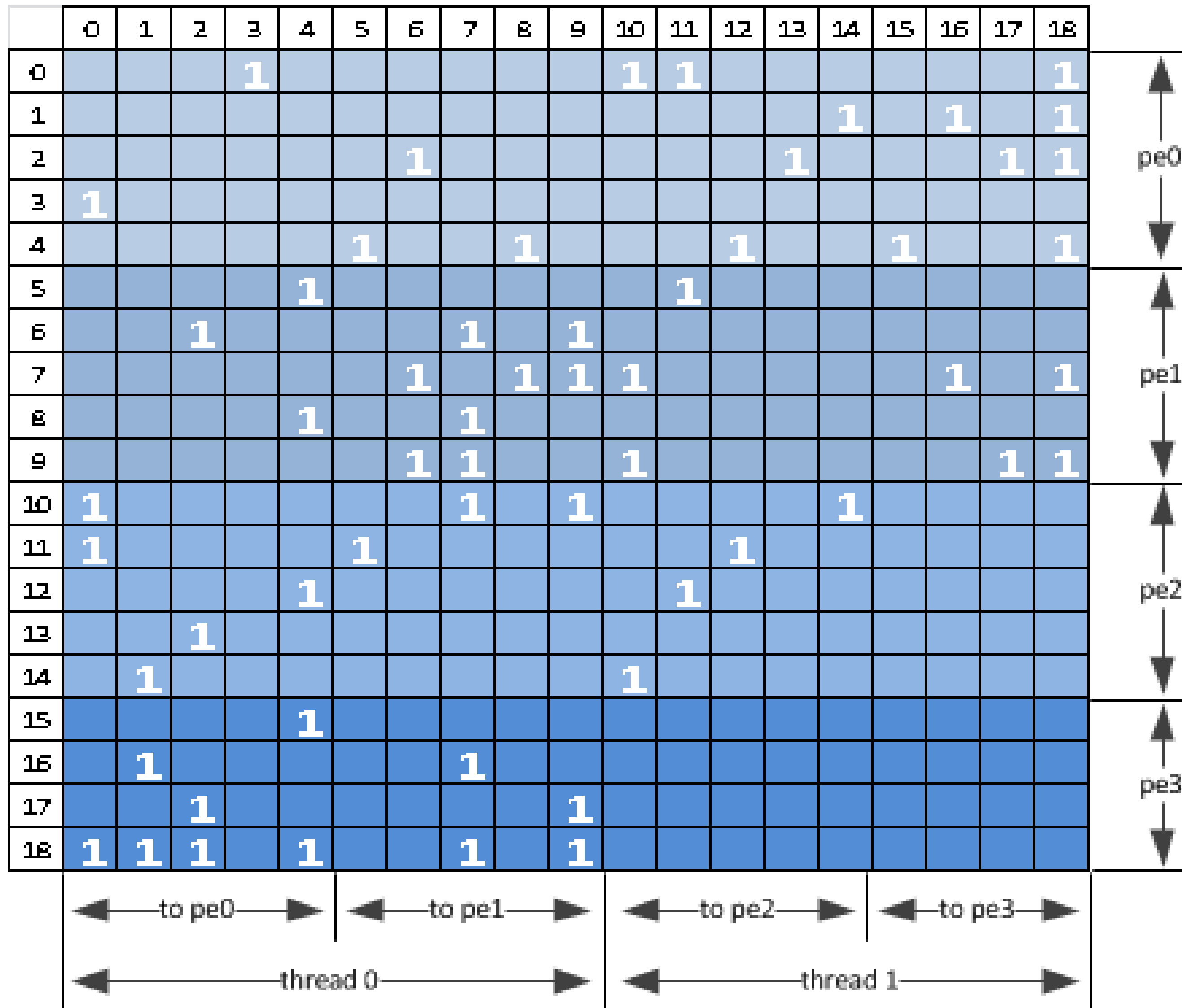
```
function ProcessQueue(Q, E)
    for all vertex ∈ Q  do
        for all w : (vertex, w) ∈ E  do
            send vertex, w to owner(w)
        end for
    end for
end function
```



thread1

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

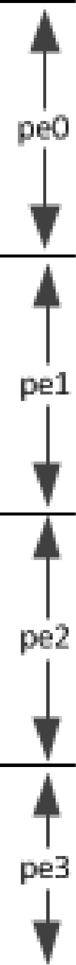
Организация параллелизма потоков



	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									

Хаотично расположенные вершины и ребра графа

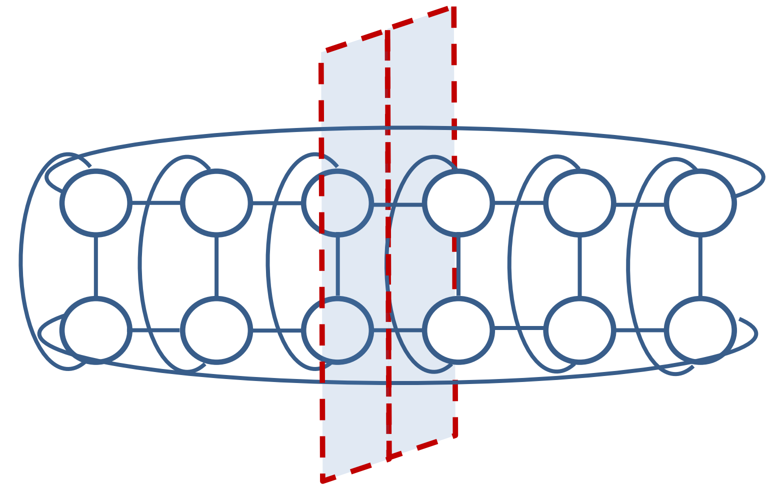
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0				1							1	1							1
1															1		1		1
2							1							1				1	1
3	1																		
4						1			1				1			1			1
5					1							1							
6			1					1		1									
7							1		1	1	1						1		1
8					1			1											
9							1	1			1							1	1
10	1				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



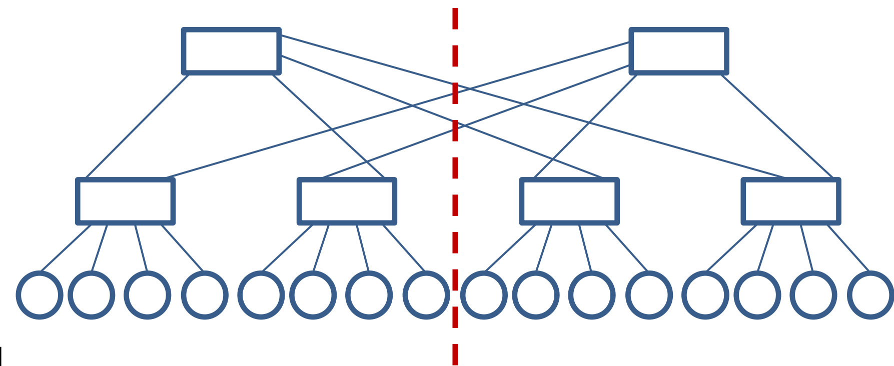
Шаблон обменов all-to-all

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 ½**
- **Посылают все узлы – для линейной масштабируемости требуется N/2 линков в бисекционной плоскости**



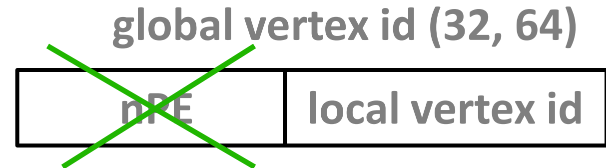
Бисекция тора = 2N/Nmax

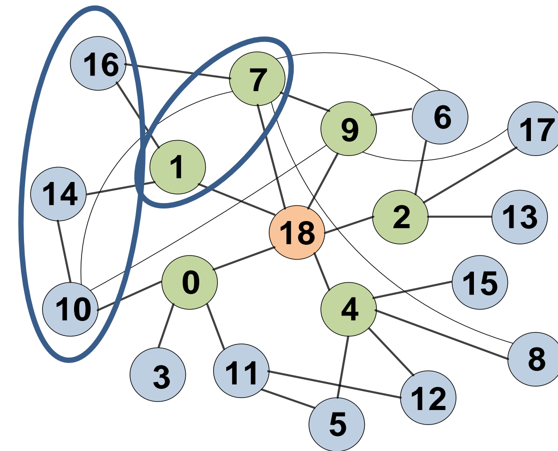


Бисекция жирного дерева (half bisection) = N/4

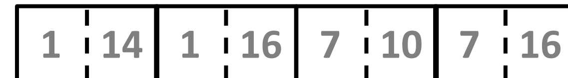
Уменьшение количества пересылаемых данных

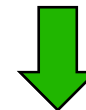
- **Использование простаивающего процессора**
- **Сокращение пересылок**
  - Отказ от лишней пересылаемой информации
  - Удаление дублирующей информации
- **Сжатие данных**
  - Использование знаний о структуре графа

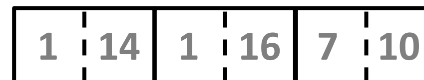




пересылаемое сообщение







Графы реального мира. Степенной закон

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

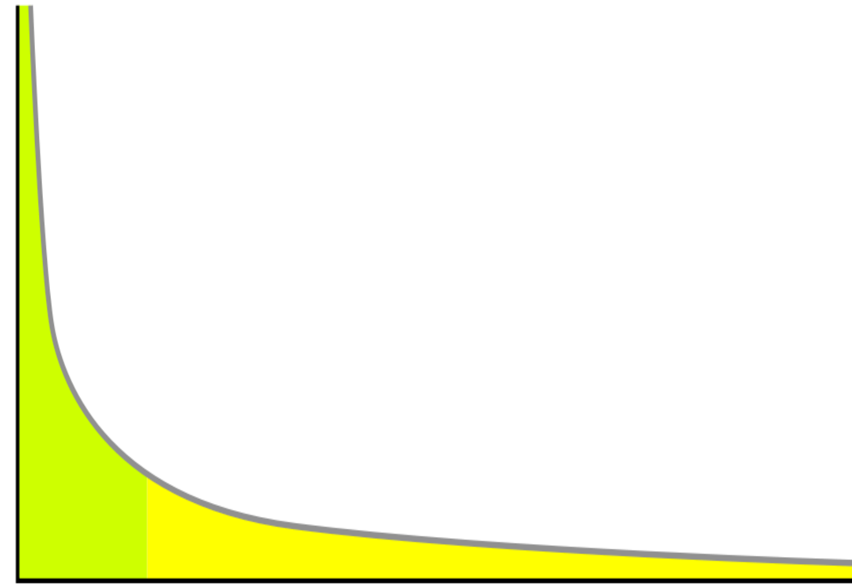
L ~ log N,

- scale-free – графы,

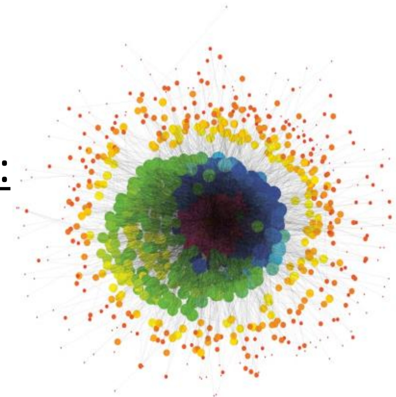
доля $P(k) \sim k^{-tau}$, $2 < tau < 3$

k – связность вершины

L ~ log log N



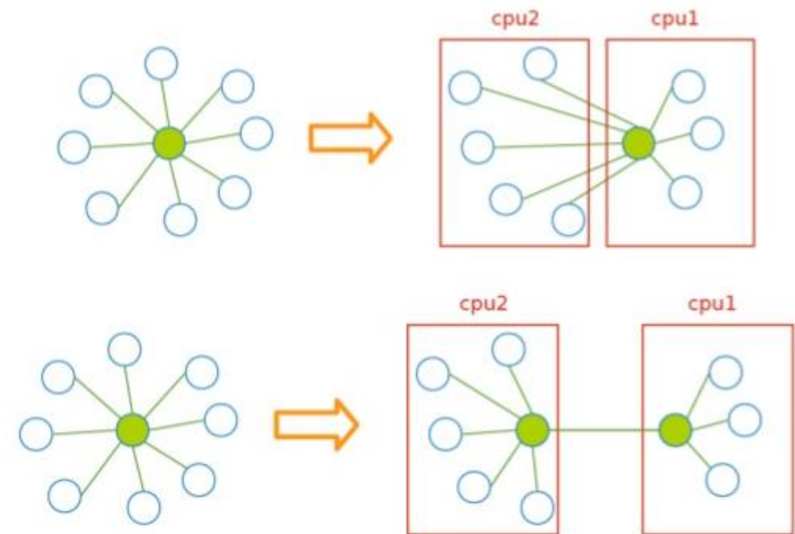
Граф Кронекера:



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
```



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