Параллельная высокопроизводительная обработка графов

ЧЕРНОСКУТОВ МИХАИЛ

ИММ УРО РАН, ИМКН УРФУ, ЕКАТЕРИНБУРГ

E-MAIL: MACH@IMM.URAN.RU

Алгоритмы на графах

Биоинформатика

Анализ социальных сетей

Бизнес-аналитка

Извлечение знаний

Планирование городской инфраструктуры

и другое…

Алгоритмы на графах

Поиск в ширину

- Простота понимания
- Широкое распространение
- Большой потенциал для распараллеливания

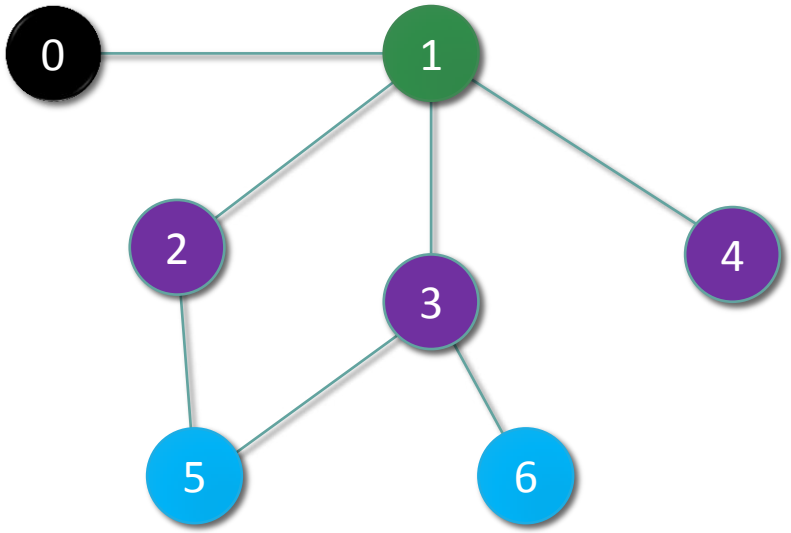
Graph500

- www.graph500.org
  - R. Murphy, K. Wheeler, B. Barrett, and J. Ang. Introducing the graph 500. In Cray User's Group (CUG), 2010
- Параллельный поиск в ширину
  - Реализации с поддержкой MPI и OpenMP
- Ориентация на обработку графов с небольшим диаметром и неравномерным распределением степеней вершин
  - "Scale-Free" графы

Параллельный поиск в ширину на графе

Синхронизированные по уровням (level synchronous) алгоритмы

- Обработка уровня N+1 начинается только после того, как закончена обработка уровня N



Препятствия для эффективной параллельной реализации

Проблема передачи данных

Проблема разметки графа

Проблема передачи данных

Описание проблемы

- Графы, используемые в практических приложениях, обладают нерегулярным шаблоном доступа к памяти
- В графах с малым диаметром значительные накладные расходы связаны с передачей данных по сети

Предлагаемое решение

- Сочетание различных типов синхронизированных по уровням алгоритмов

Синхронизированные по уровням алгоритмы

Прямой обход графа (top-down)

- Традиционный подход к реализации поиска в ширину на графе
- Активная на текущей итерации родительская вершина помечает всех своих соседей

Обратный обход графа (bottom-up)

- Неактивные вершины на текущей реализации пытаются отыскать активные вершины среди своих соседей
- S. Beamer, K. Asanovic, D. A. Patterson, Direction-optimizing breadth-first search // in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), 2012.

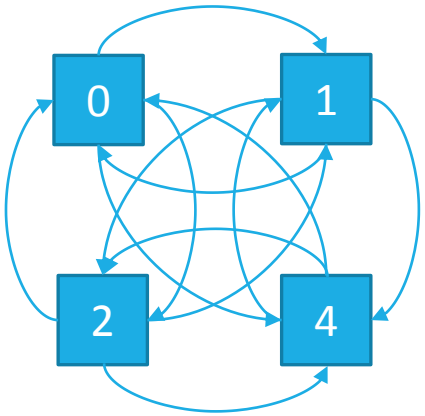
Прямой обход графа

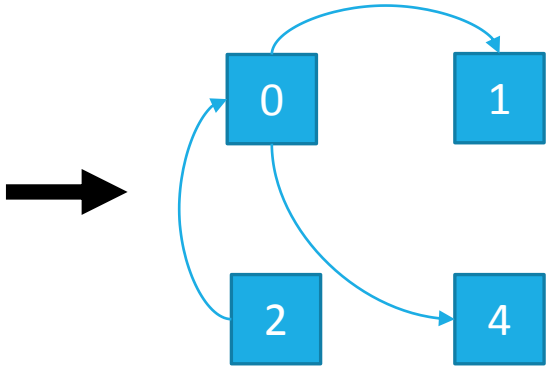
```
for all u in dist
    dist[u] ← -1
dist[s] ← 0
level ← 0
do

    parallel for each vert in V.this_node
        if dist[vert] = level
            for each neighb in vert.neighbors
                if neighb in V.this_node
                    if dist[neighb] = -1
                        dist[neighb] ← level + 1
                        pred[neighb] ← vert
                else
                    vert_batch_to_send.push(neighb)

    send(vert_batch_to_send)
    receive(vert_batch_to_receive)

    parallel for each vert in vert_batch_to_receive
        if dist[vert] = -1
            dist[vert] ← level + 1
            pred[vert] ← vert.pred
    level++
while(!check_end())
```

Прямой обход графа

Обмены данными производятся с помощью матрицы коммуникаций

- a_{ij} – количество данных (в байтах), которые необходимо передать от узла i к узлу j



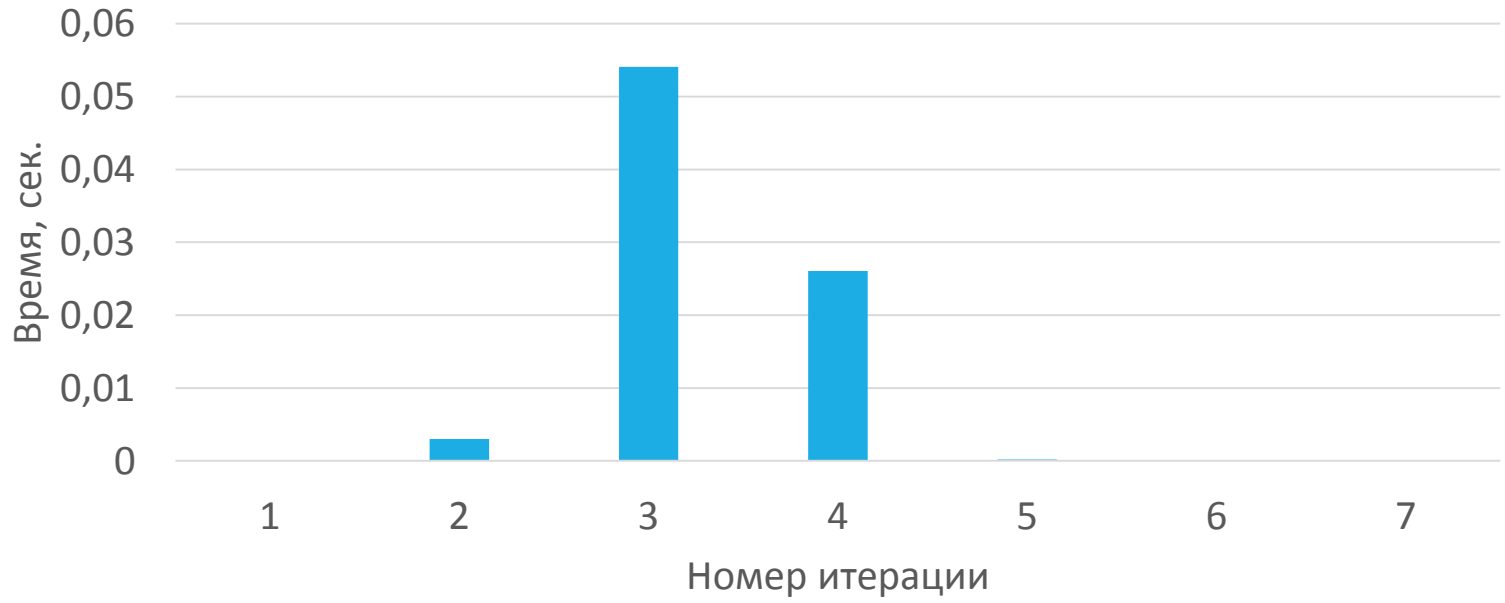


$$\begin{pmatrix} 0 & 10 & 0 & 20 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Прямой обход графа

Время, затраченное на обмены данными на каждой итерации алгоритма

- Граф с ~1 млн. вершин, распараллеливание на 4 выч. узла
- Общее время, затраченное на обмены данными – 0,83 сек.

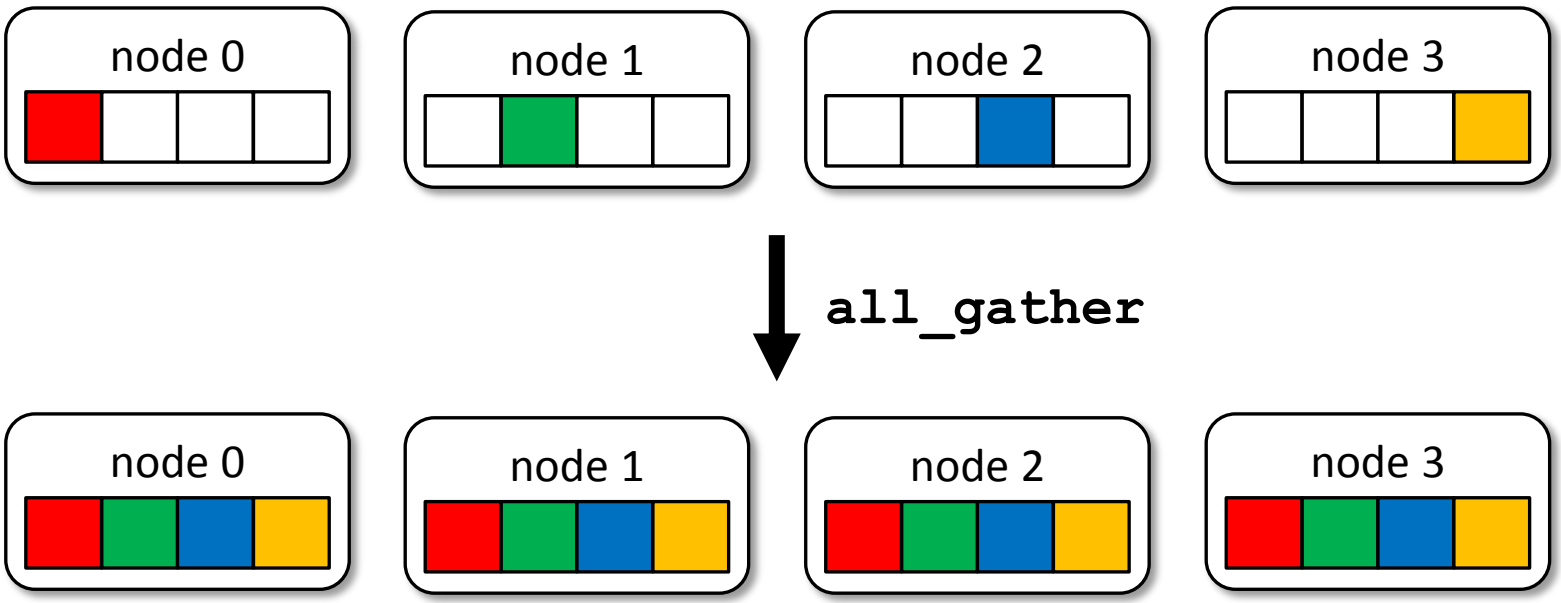


Обратный обход графа

```
for all u in dist
    dist[u] ← -1
dist[s] ← 0
level ← 0
do
    parallel for each vert in V.this_node
        if dist[vert] = -1
            for each neighb in vert.neighbors
                if bitmap_current.neighb = 1
                    dist[vert] ← level + 1
                    pred[vert] ← neighb
                    bitmap_next.vert ← 1
                    break

    all_gather(bitmap_next)
    swap(bitmap_current, bitmap_next)

    level++
while(!check_end())
```

Обратный обход графа

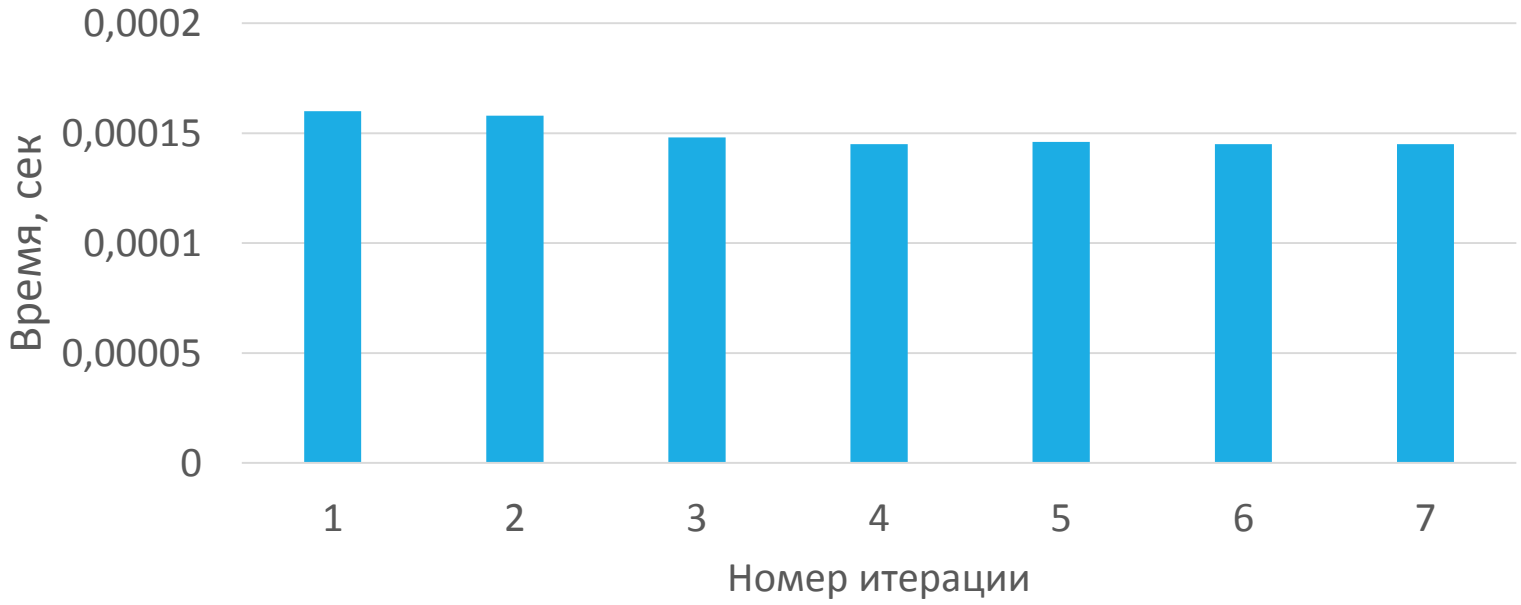
Синхронизация данных происходит посредством выполнения коллективных операций



Обратный обход графа

Время, затраченное на обмены данными на каждой итерации алгоритма

- Граф с ~1 млн. вершин, распараллеливание на 4 узла
- Общее время, затраченное на обмены данными – 0,001 сек.



Проблема передачи данных

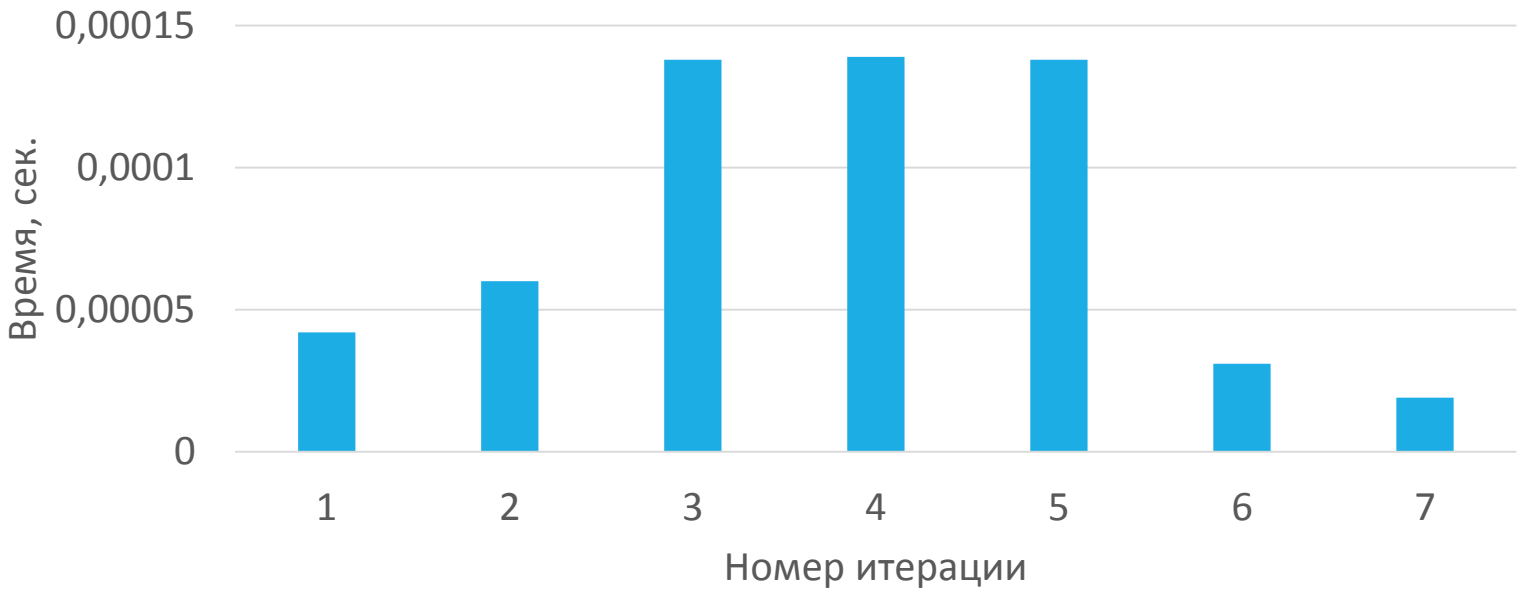
Предлагаемое решение – гибридный обход графа

- Первые две итерации – “top-down”
- Следующие три итерации – “bottom-up”
- Остальные итерации – “top-down”

Гибридный обход графа

Время, затраченное на обмены данными на каждой итерации алгоритма

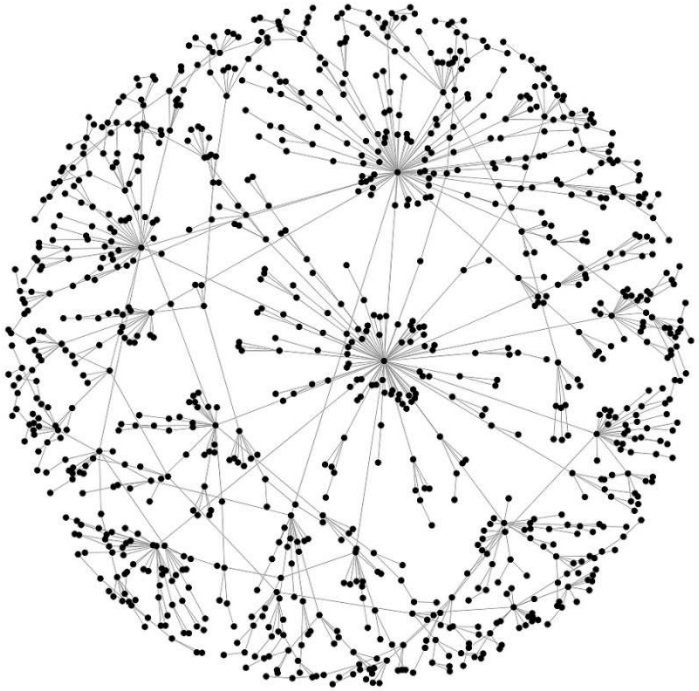
- Граф с ~1 млн. вершин, распараллеливание на 4 узла
- Общее время, затраченное на обмены данными – 0,0005 сек.



Проблема разметки графа

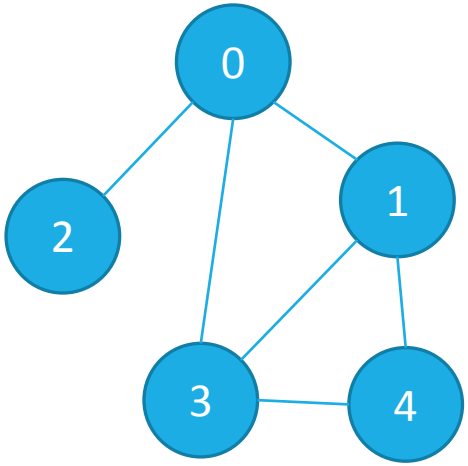
Неравномерное распределение степеней

- Много вершин с малым числом входящих / исходящих ребер
- Мало вершин с большим числом входящих / исходящих ребер



Проблема разметки графа

Наиболее распространенный формат хранения данных о внутренней структуре графа – CSR



Row pointers: 0, 3, 6, 7, 10, 12

Column ids: 1, 2, 3, 0, 3, 4, 0, 0, 1, 2, 1, 3

Проблема разметки графа

Использование формата CSR предполагает обход массива **Row pointers**

- Заранее неизвестно, сколько ребер инцидентны данной вершине (т.е. сколько элементов в массиве **Column ids** нужно просмотреть)

В синхронизированных по уровням алгоритмах скорость выполнения текущей итерации определяется скоростью обработки самой “тяжелой” вершины

- Дисбаланс нагрузки

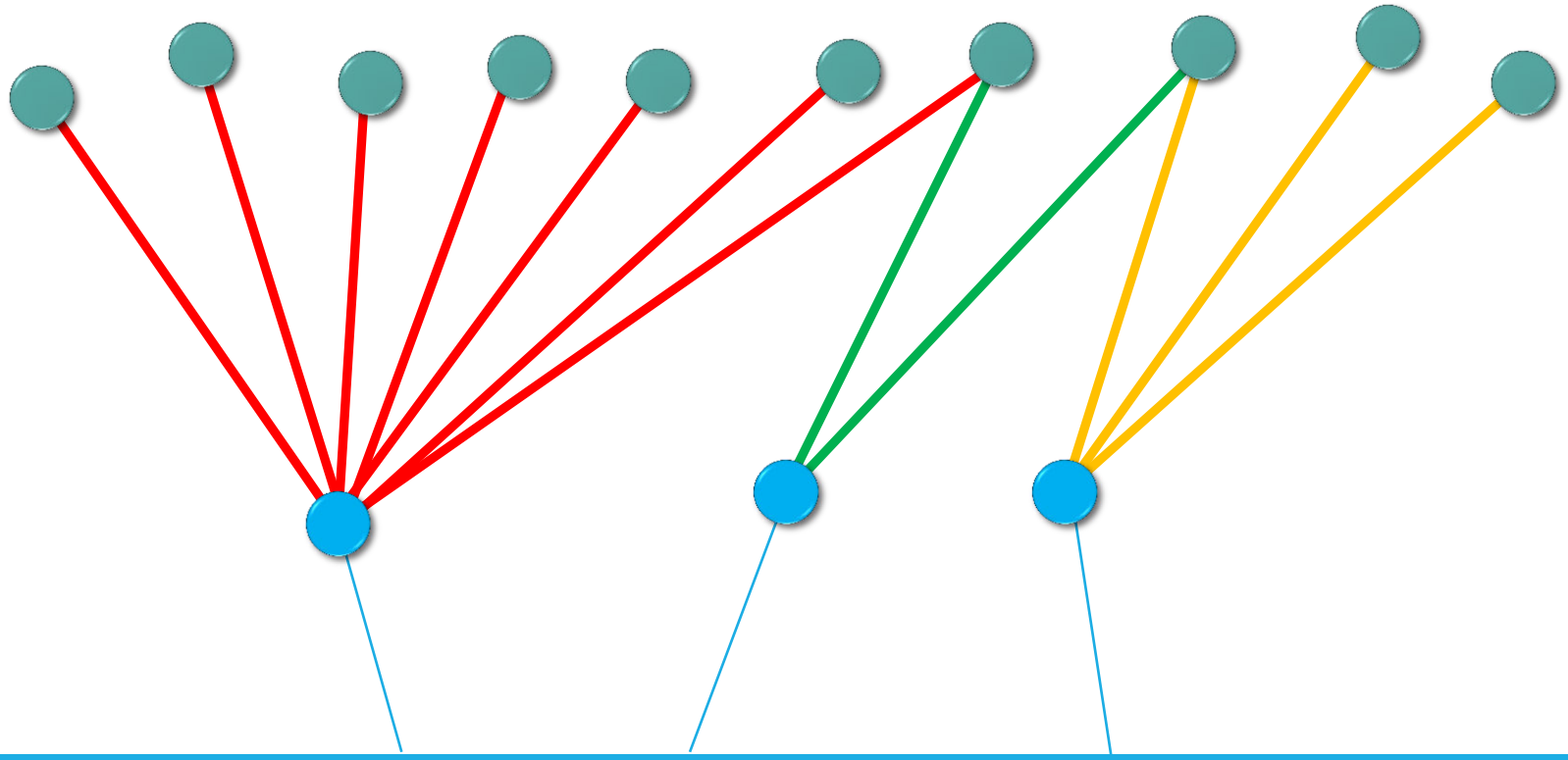
Проблема разметки графа

Предлагаемое решение – метод балансировки нагрузки

- Разделим массив **Column ids** на равные части величиной **Max edges** элементов
- Установим соответствие между каждой частью массива **Rowstarts** и массива **Column ids** с помощью вспомогательного массива **Part column**
- Каждый поток будет обрабатывать ограниченное количество ребер из соответствующего интервала, определяющегося с помощью элементов массива **Part column**

Проблема разметки графа

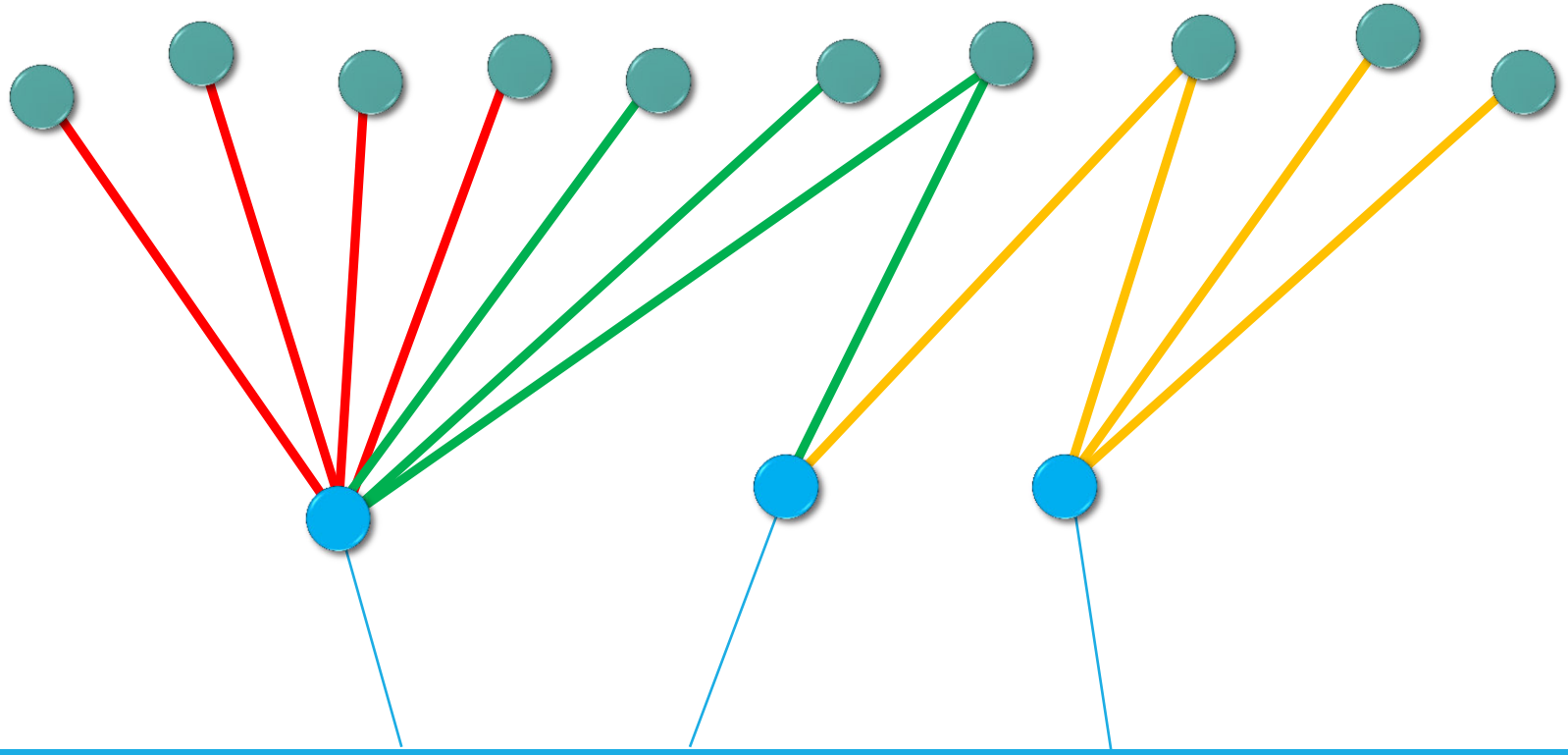
Обработка графа без балансировки нагрузки



Проблема разметки графа

Обработка графа без балансировки нагрузки

- `max_edges = 4`



Проблема разметки графа

Разметка массива **Part column**

```
parallel for i in V.this_node
    first ← V.this_node[i]
    last ← V.this_node[i+1]
    index ← round_up(first/max_edges)
    current ← index*max_edges
    while(current < last)
        part_column[index] ← i
        current += max_edges
        index++
```

Прямой обход графа

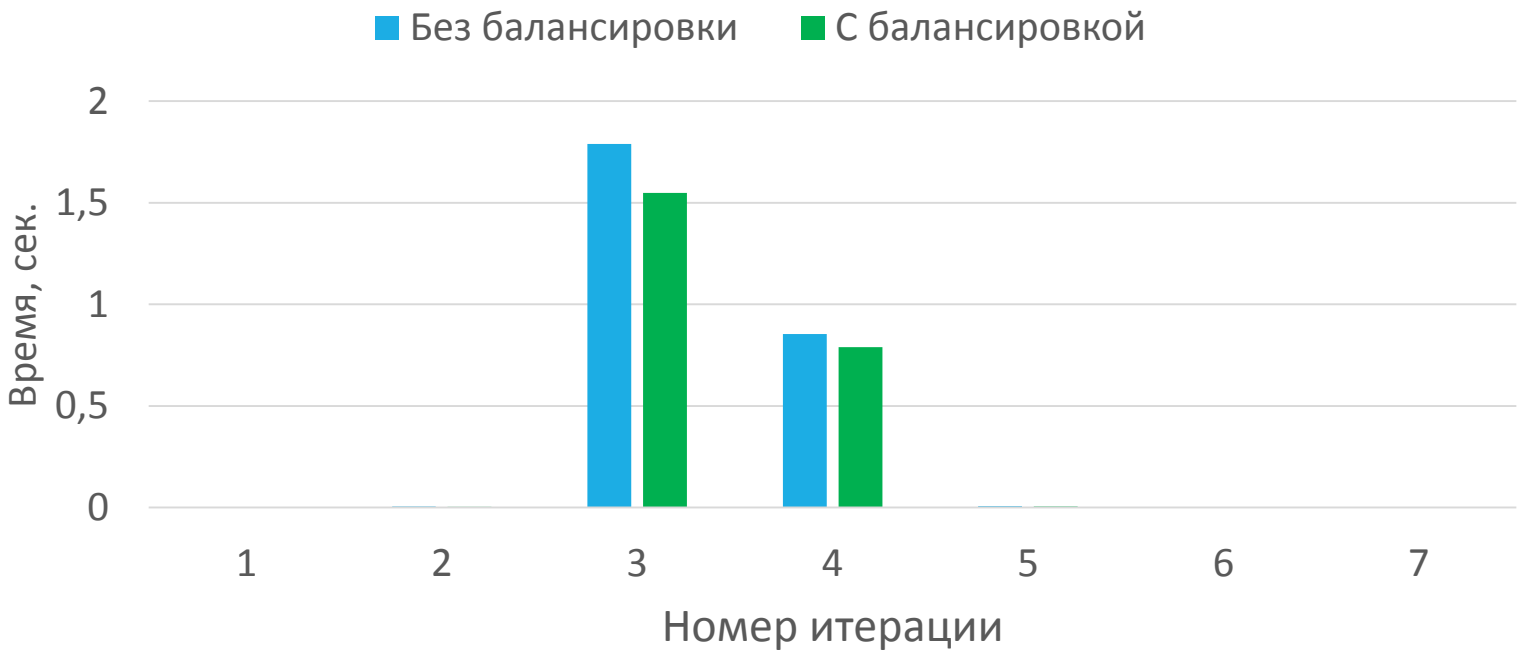
Главный цикл (разметка вершин) синхронизированного по уровням поиска в ширину, модифицированный для балансировки нагрузки

```
// preparation...
parallel for i in part_column
    first_edge ← i*max_edges
    last_edge ← (i+1)*max_edges
    curr_vert ← part_column[i]
    for each edge ϵ [first_edge;last_edge)
        if neighbors of curr_vert ϵ
         [first_edge;last_edge)
            if dist[curr_vert] = level
            for each k ϵ neighbors of curr_vert
            if dist[k] = -1
                dist[k] ← level + 1
                pred[k] ← curr_vert
        curr_vert++
// data synchronization...
```

Прямой обход графа

Время работы разметки графа на каждой итерации алгоритма без балансировки нагрузки и с балансировкой нагрузки

- Граф с ~1 млн. вершин, распараллеливание на 4 узла



Обратный обход графа

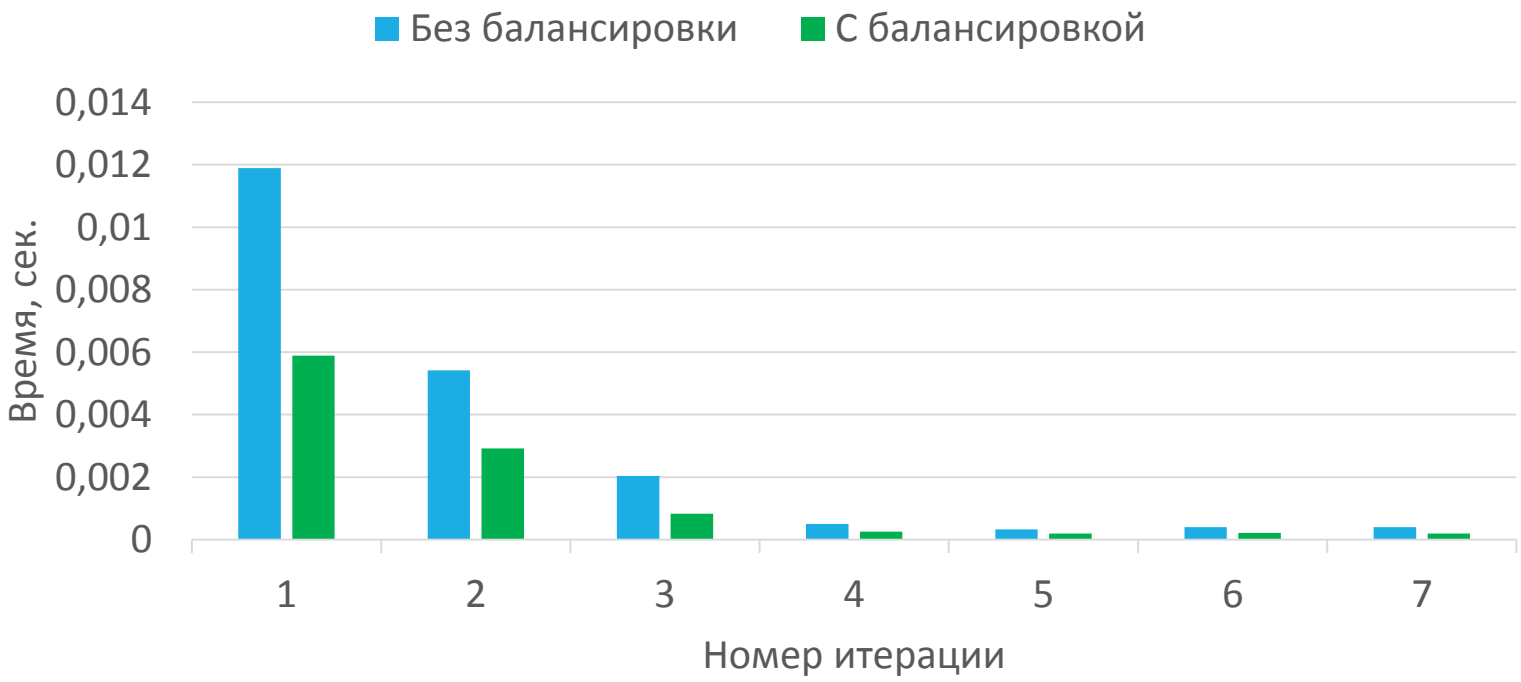
Главный цикл (разметка вершин) синхронизированного по уровням поиска в ширину, модифицированный для балансировки нагрузки

```
// preparation...
parallel for i in part_column
    first_edge ← i*max_edges
    last_edge ← (i+1)*max_edges
    curr_vert ← part_column[i]
    for each edge ϵ [first_edge;last_edge)
        if neighbors of curr_vert ϵ
         [first_edge;last_edge)
            if dist[curr_vert] = -1
                for each k ϵ neighbors of curr_vert
                    if bitmap_current.k = 1
                        dist[curr_vert] ← level + 1
                        pred[curr_vert] ← k
                        bitmap_next.vert ← 1
                            break
        curr_vert++
// data synchronization...
```

Обратный обход графа

Время работы разметки графа на каждой итерации алгоритма без балансировки нагрузки и с балансировкой нагрузки

- Граф с ~1 млн. вершин, распараллеливание на 4 узла



Сочетание методов

Разработанные методы могут быть использованы совместно для максимального ускорения параллельного алгоритма поиска в ширину на графе

- Метод балансировки нагрузки – для снижения времени разметки графа на каждой итерации
- Метод гибридизации обхода – для оптимизации этапа синхронизации данных в конце каждой итерации алгоритма

Тестирование

Разработанные методы встроены в собственную (custom) реализацию теста Graph500

Измерим производительность custom-реализации для разного количества узлов

- 1, 2, 4, 8 узлов кластера “Уран”
  - CPU Intel Xeon X5675, ОЗУ 192 ГБ
- Параметр “Scale” варьируется от 20 до 25

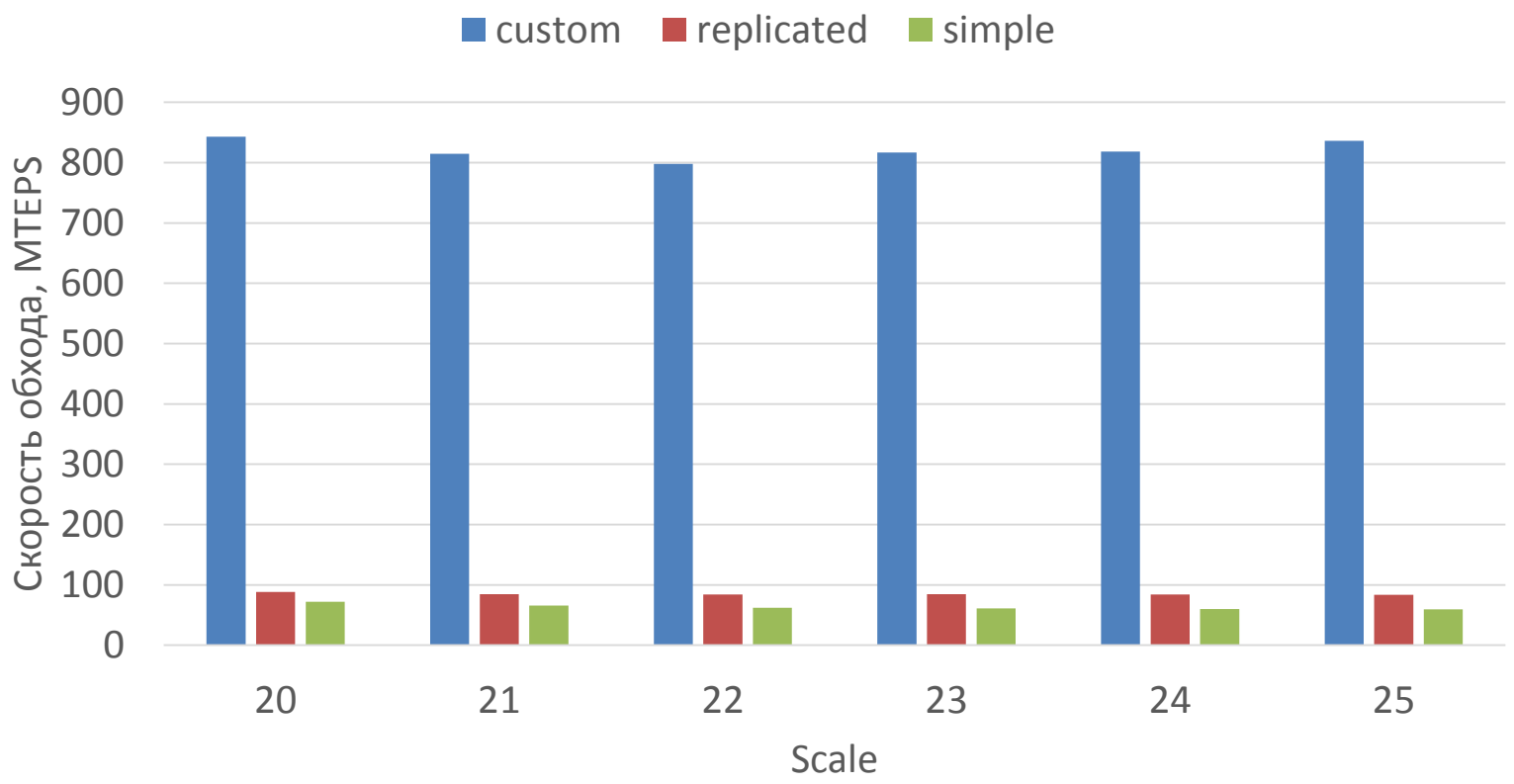
Сравним custom-реализацию со стандартными реализациями, предоставленными оргкомитетом Graph500

- Simple-реализация
- Replicated-реализация

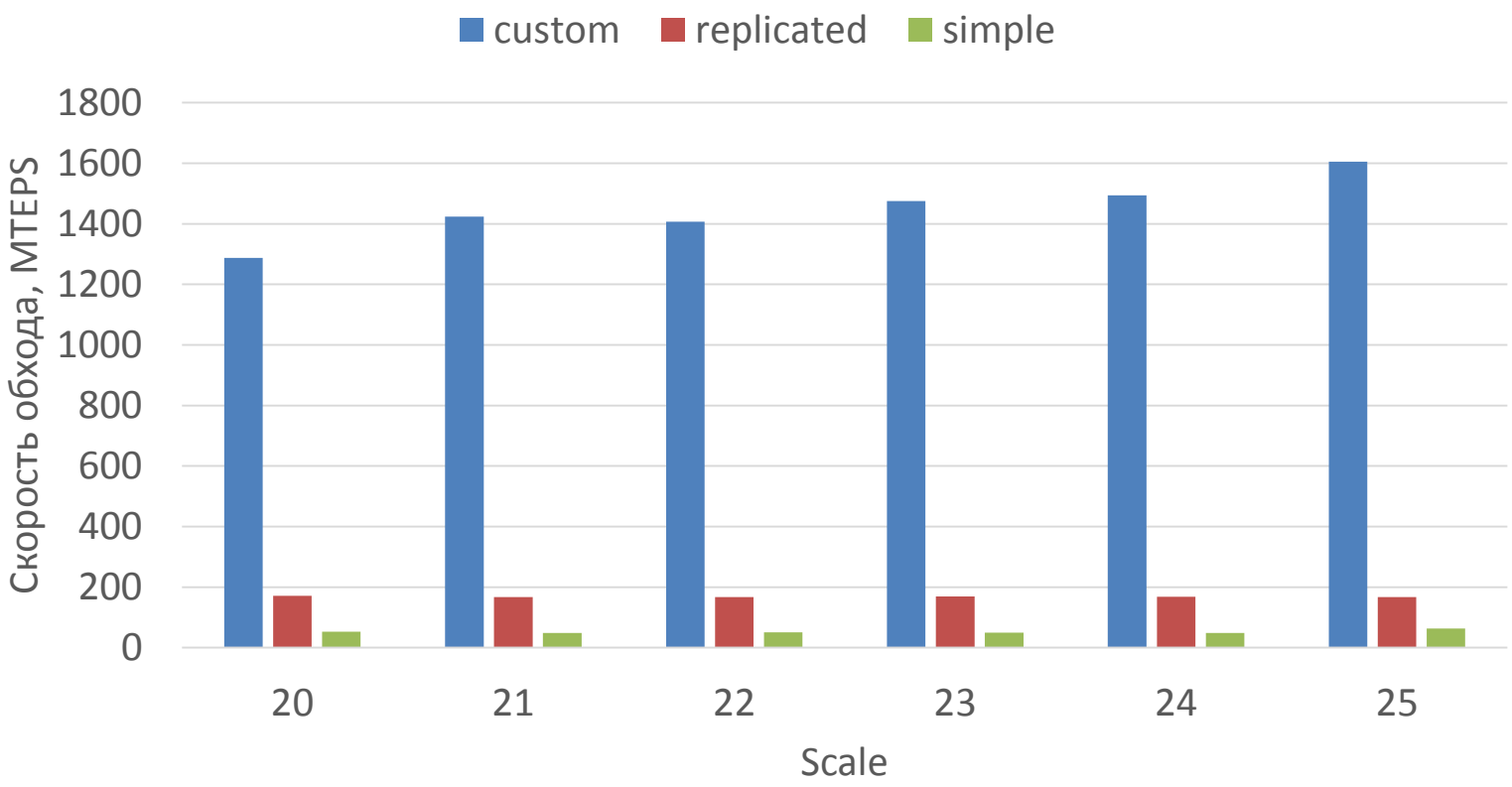
Основной показатель эффективности – скорость обхода графа

- Измеряется в количестве пройденных в секунду ребер (Traversed Edges Per Second, TEPS)

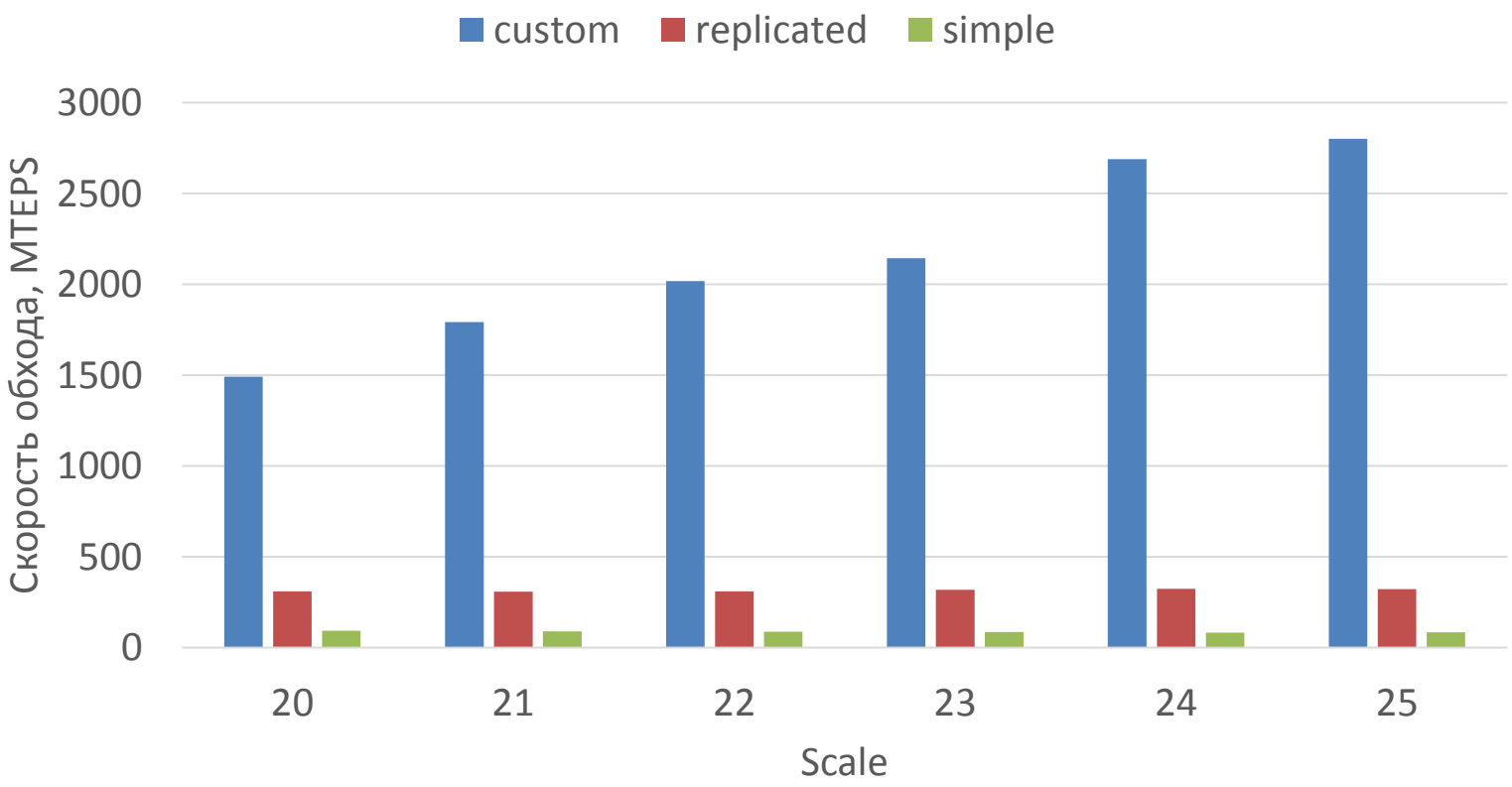
Результаты (1 узел)



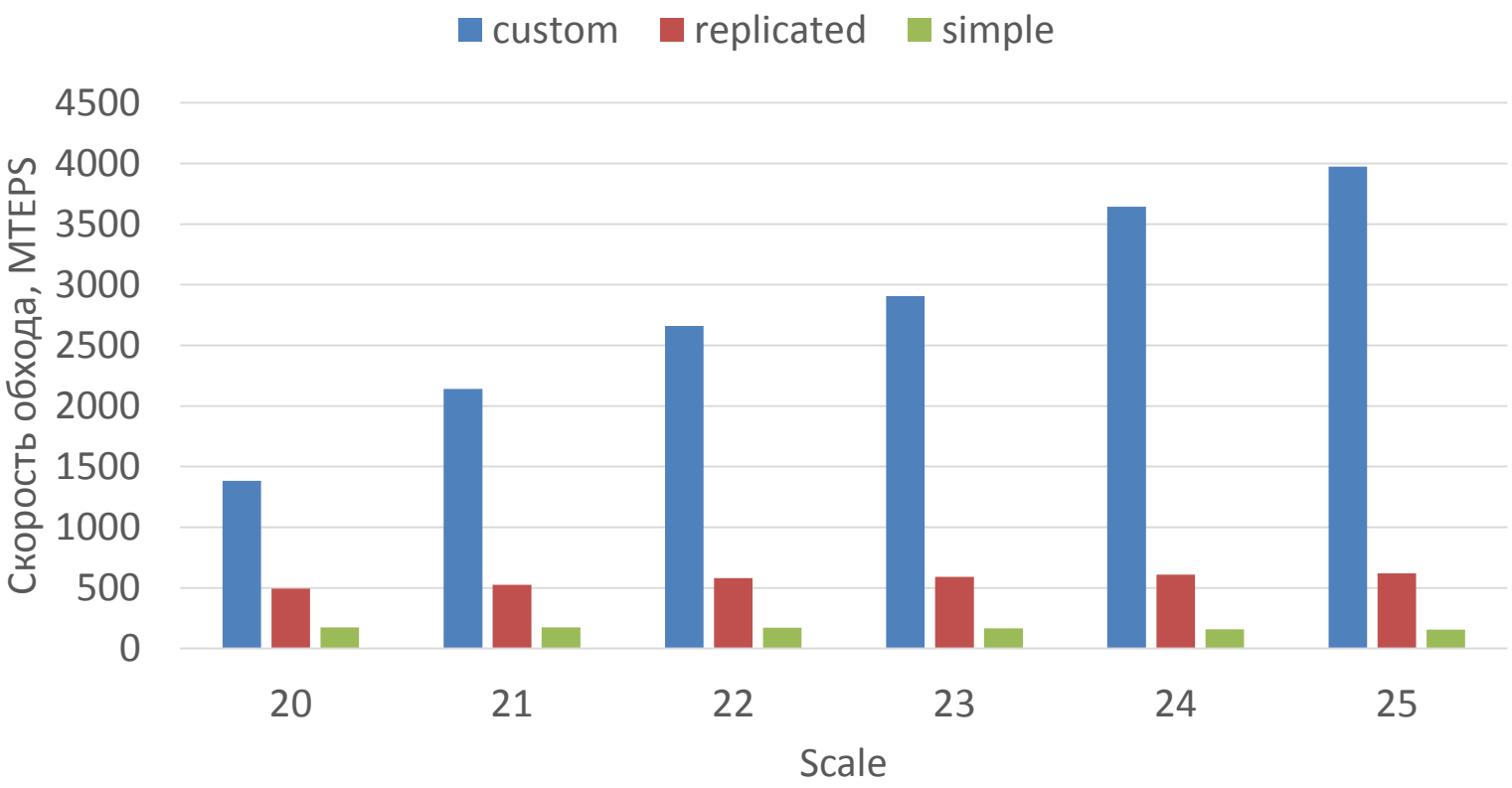
Результаты (2 узла)



Результаты (4 узла)



Результаты (8 узлов)



Результаты

Сочетание метода балансировки нагрузки и метода гибридизации обхода помогает добиться значительного ускорения параллельного поиска в ширину на графе

Custom-реализация сохраняет потенциал масштабируемости

Итоги

Метод балансировки нагрузки помогает снизить накладные расходы на разметку графа в алгоритме параллельного поиска в ширину

Метод гибридизации обхода графа помогает сделать этап синхронизации данных на каждой итерации алгоритма параллельного поиска в ширину более эффективным

Планы на будущее

- Изучить масштабируемость предложенного решения
- Модифицировать custom-реализацию для использования ускорителей вычислений и сопроцессоров

Вопросы?