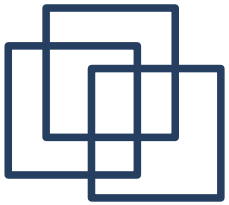
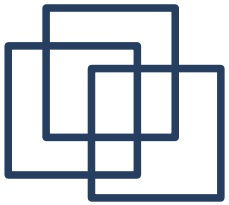


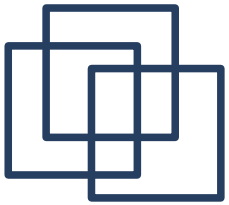
Реализация и исследование теста производительности Graph500 для многоузловых систем с GPU

Черноскутов Михаил
ИММ УрО РАН, УрФУ
mach@imm.uran.ru



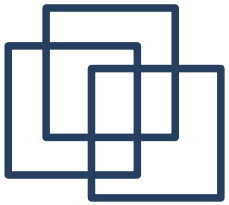
Содержание

- Data intensive computing
- Graph500
- multiGPU Graph500



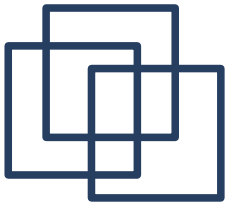
Суперкомпьютерные вычисления

- Computational intensive
  - «Вычислительно-интенсивные»
  - Большое количество вычислительных операций
  - Регулярный шаблон доступа к памяти
- Data intensive
  - «С интенсивным доступом к данным»
  - Количество вычислительных операций невелико
  - Нерегулярный шаблон доступа к памяти



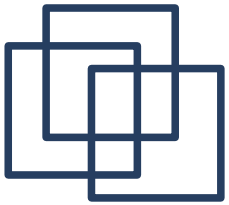
Актуальность

- Серийные комплектующие ориентированы на решение задач класса «Computational Intensive»
- Задачи класса «Data Intensive» все чаще появляются в приложениях промышленного и научного секторов
- Противоречие
  - Серийное аппаратное обеспечение + новые задачи = ?



Актуальность

- Требуется количественная оценка, показывающая эффективность решения задач класса «Data Intensive»
  - Насколько эффективна та или иная архитектура?
  - Есть ли предел масштабирования?
- Решение
  - Провести тестирование вычислительной системы с помощью задачи класса «Data Intensive»

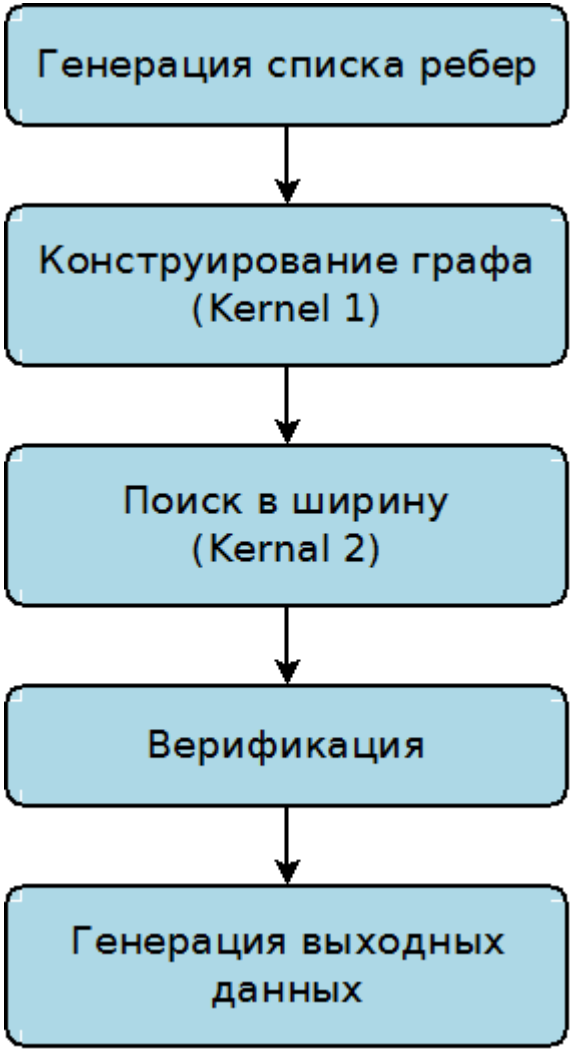


Graph500

- Вычислительное ядро
  - Параллельный поиск в ширину на большом графе
- Предпосылки
  - Широкое распространение алгоритма
  - Корреляция результатов с реальными приложениями
  - Возможность использования близких к реальным наборов входных данных

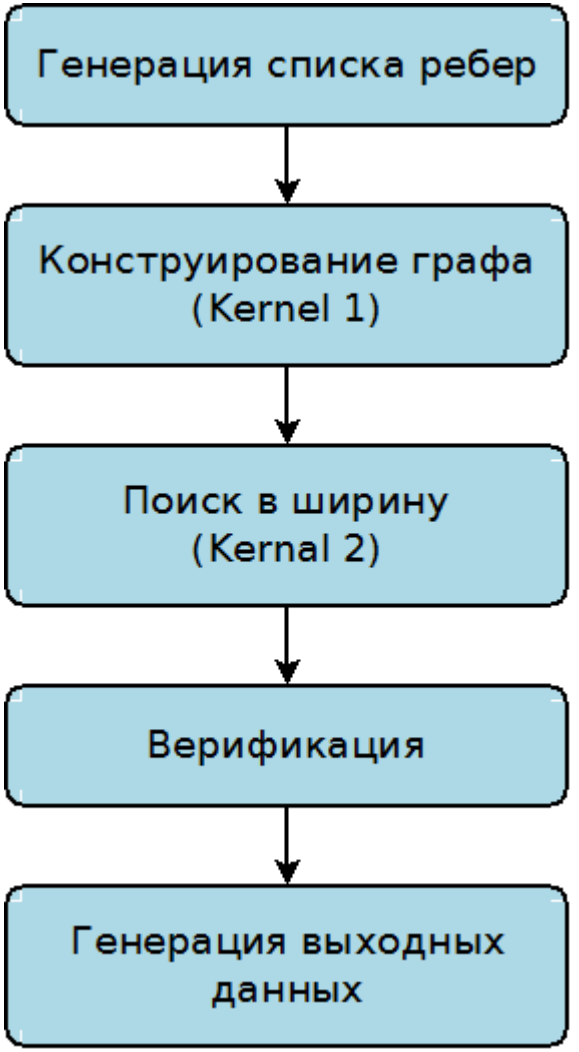
Структура теста

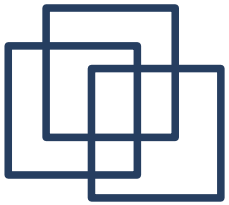
- Набор тестов
  - 4 reference-реализации
    - simple
    - replicated
    - replicated_csc
    - one_sided
  - Шаблон для custom-реализации



Структура теста

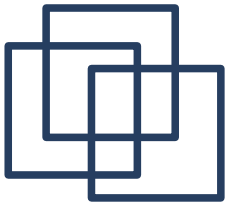
- Входные данные
  - SCALE: логарифм по основанию 2 от числа вершин
  - Edgefactor: отношение количества ребер к количеству вершин (16 по умолчанию)
- Выходные данные
  - Скорость обхода графа в TEPS (Traversed Edges Per Second)
  - Статистическая информация





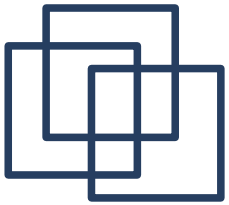
СКЦ ИММ УрО РАН

- СК «Уран»
  - Вычислительные узлы
    - 160 серверов HP ProLiant BL220c
    - 30 серверов HP ProLiant SL390s G7
      - 8 GPU Nvidia Tesla M2050/M2090
    - 16 серверов HP ProLiant SL270s Gen8
      - 8 GPU Nvidia Tesla M2090
  - Интерконнект
    - 20 Гб/с



СКЦ ИММ УрО РАН

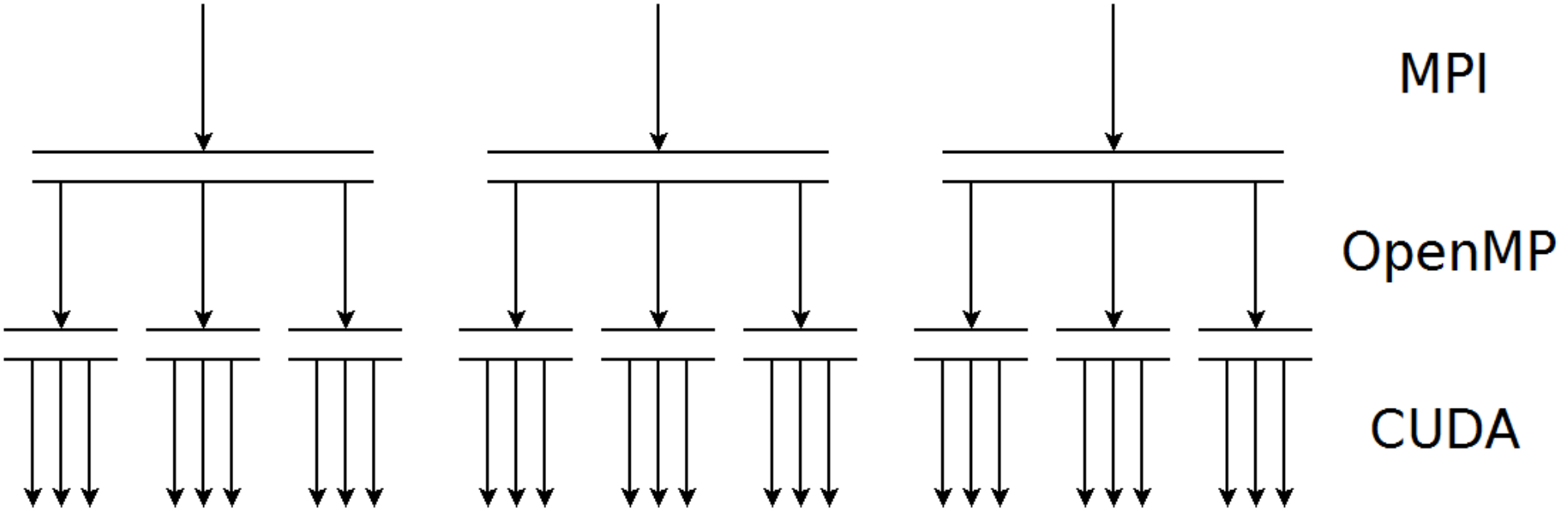
- СК «Уран» в рейтингах
  - Top500
    - 428-е место в редакции от июня 2013 г.
  - Top50
    - 6-е место в редакции от 24.09.2013 г.
  - Green500
    - 385-е место в редакции от июня 2013 г.
  - Graph500
    - 104-е место в редакции от ноября 2013 г.
    - reference-реализация



Custom Graph500

- Цель
  - Разработать собственную реализацию теста производительности Graph500, ориентированную для запуска на гибридных вычислительных системах с GPU-ускорителями вычислений
- Направленность
  - Развитие методов эффективной загрузки гибридных вычислительных систем в задачах класса «Data Intensive»
  - Улучшение результатов СКЦ ИММ УрО РАН в рейтинге Graph500

Схема распараллеливания



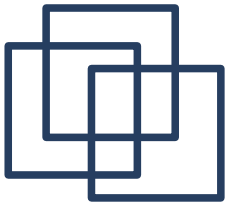


Схема распараллеливания

- CUDA
  - Обработка отдельных вершин/ребер в графе
- OpenMP
  - Обмен данными между CPU и GPU
- MPI
  - Обмен данными между узлами

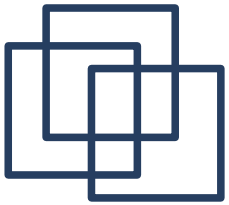
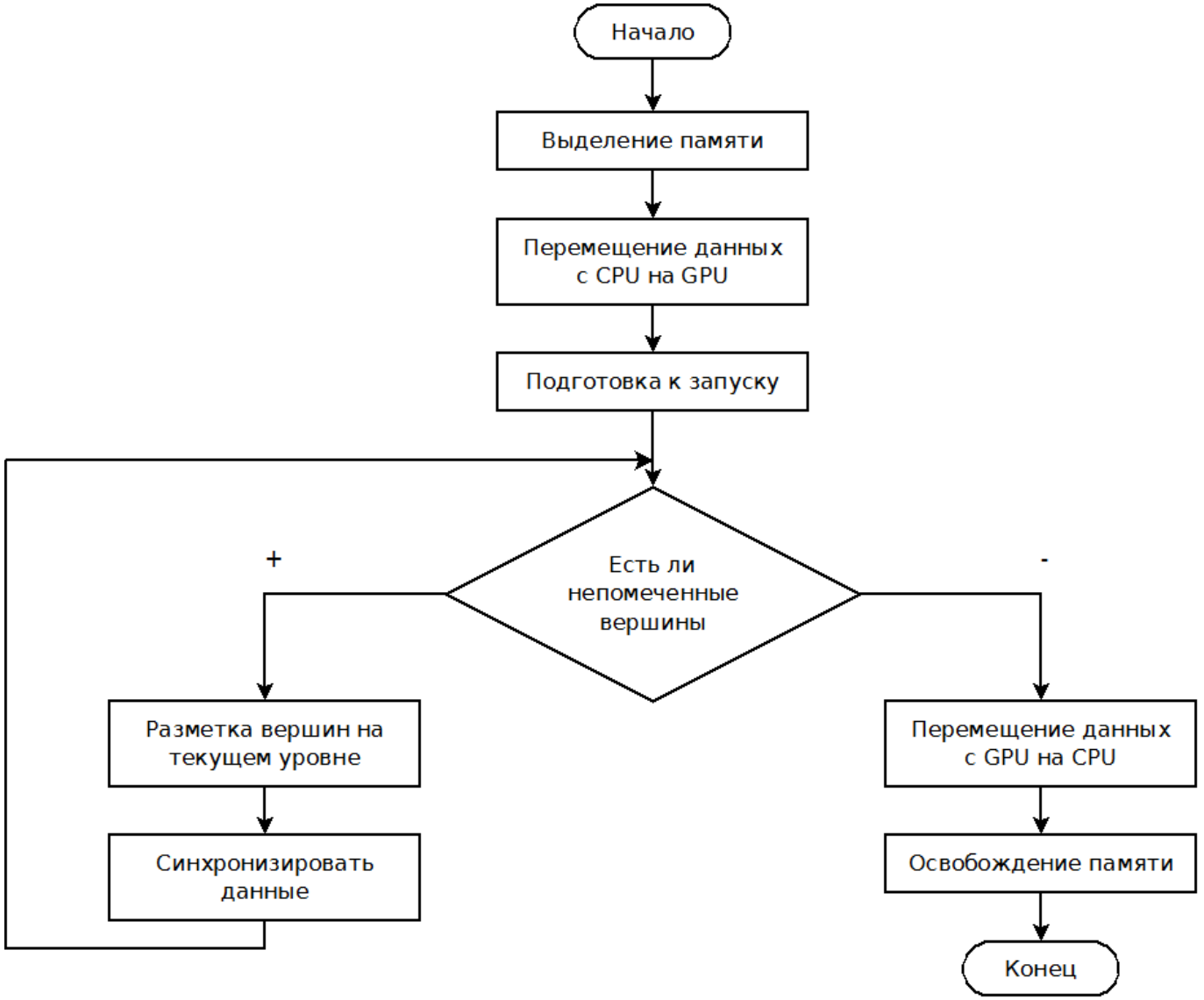
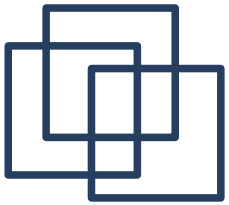


Схема распараллеливания

- Особенности
  - Мелкозернистый параллелизм
  - «Сужение» пропускной способности канала передачи данных на пути от GPU к интерконнекту
- Препятствия
  - Дисбаланс нагрузки
  - Синхронизация данных

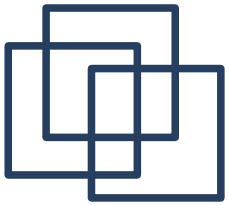
Схема распараллеливания





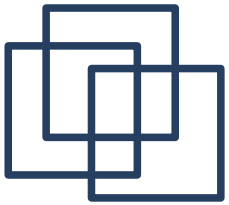
Синхронизированные по уровням алгоритмы

- Обход уровня (N+1) начинается только после полного исследования уровня N
  - На основе очередей
  - На основе обхода



Синхронизированные по уровням алгоритмы

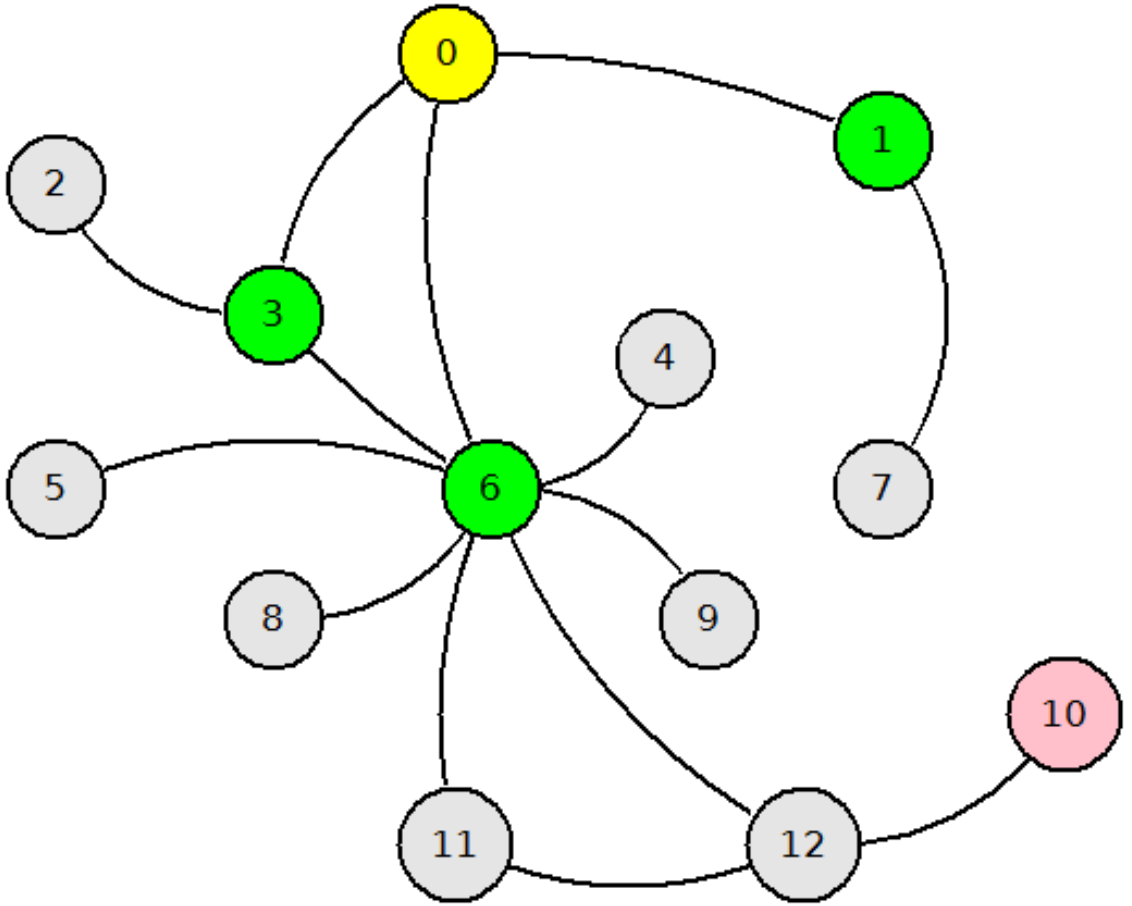
- На основе очередей
  - Использование очереди для хранения обрабатываемых на текущей итерации вершин
  - Атомарное добавление и извлечение вершин из очереди
  - Плохо подходит для модели программирования GPU



Синхронизированные по уровням алгоритмы

- На основе обхода
  - Полный обход всех вершин на каждой итерации
  - Использование атомарных операций сведено до минимума
  - Квадратичная сложность
  - Хорошо подходит для модели программирования GPU

Анализ алгоритма

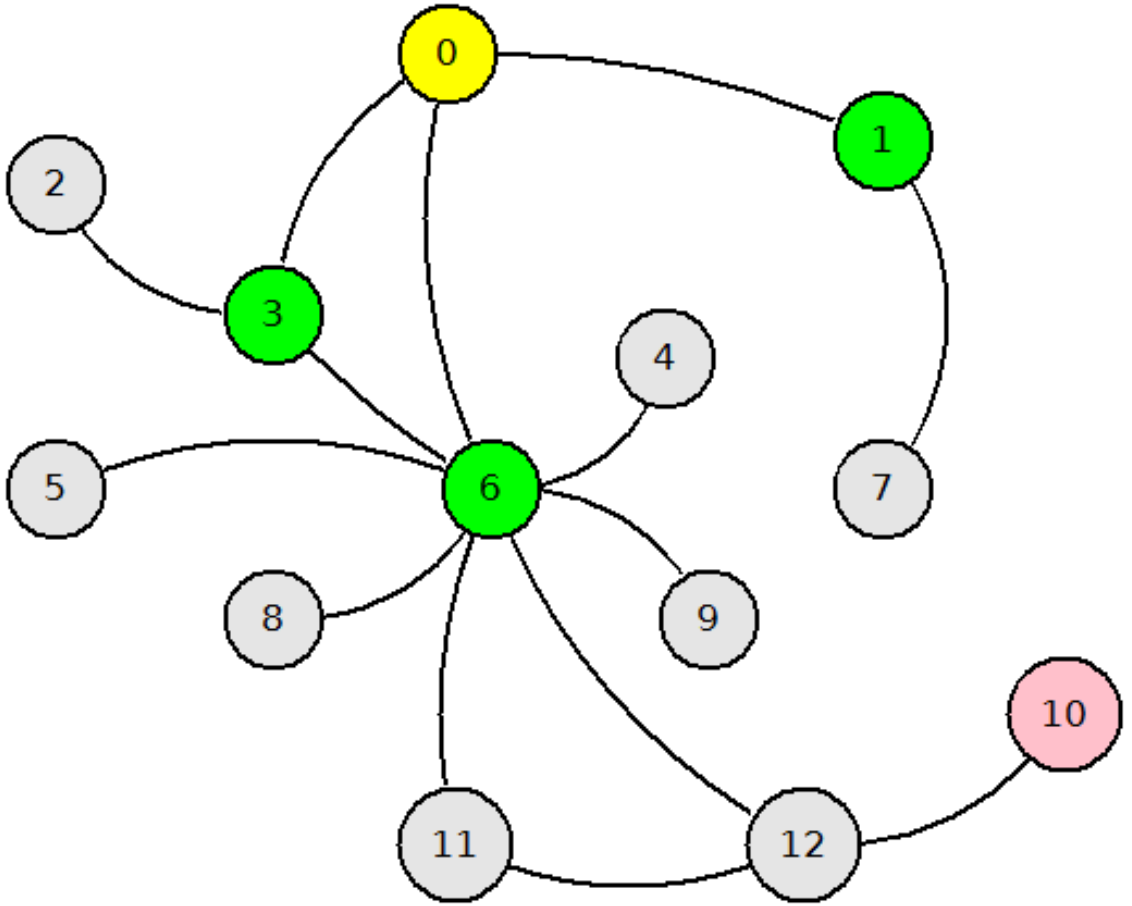


Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

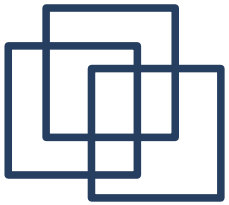
Анализ алгоритма



Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

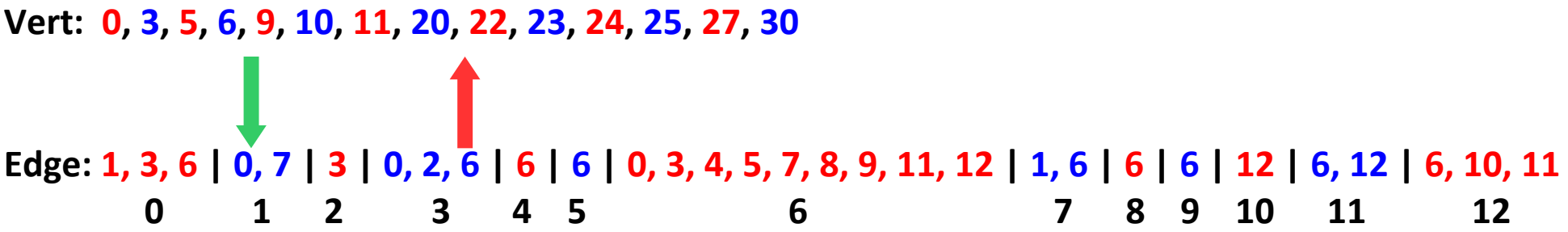
Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

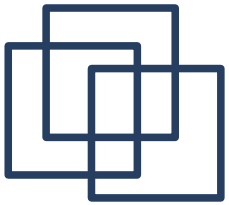
0 1 2 3 4 5 6 7 8 9 10 11 12



Анализ алгоритма

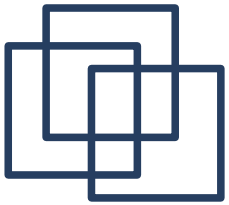
- Как увеличить производительность?
  - Снизить дисбаланс нагрузки
- Какие препятствия?
  - Обход вершин, а не ребер
  - Невозможно определить номер вершины по номеру элемента в массиве Edge





Метод балансировки нагрузки

- Разделим массив Edge на равные части
- Установим соответствие между каждой частью массива Edge и массива Vert с помощью массива Start_Vert
- Каждый поток будет обрабатывать ограниченное количество ребер из соответствующего интервала



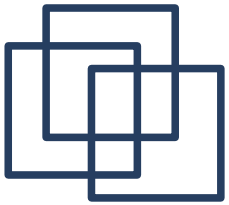
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: x, x, x, x, x, x, x, x



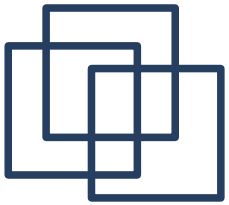
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, x, x, x, x, x, x



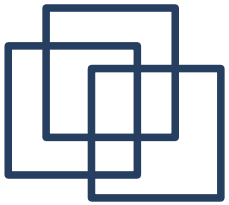
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, 1, x, x, x, x, x



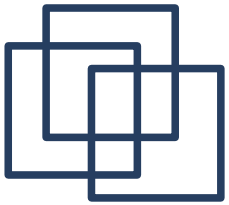
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, 1, 3, x, x, x, x, x



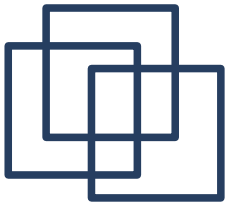
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, 1, 3, 6, x, x, x, x



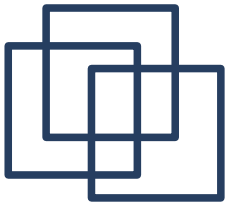
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, 1, 3, 6, 6, x, x, x



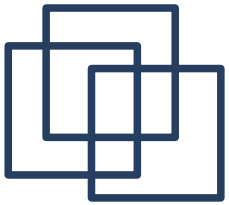
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12

Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



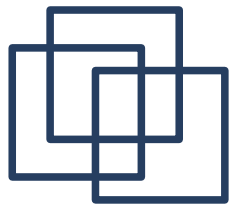
Заполнение массива Start_Vert

Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

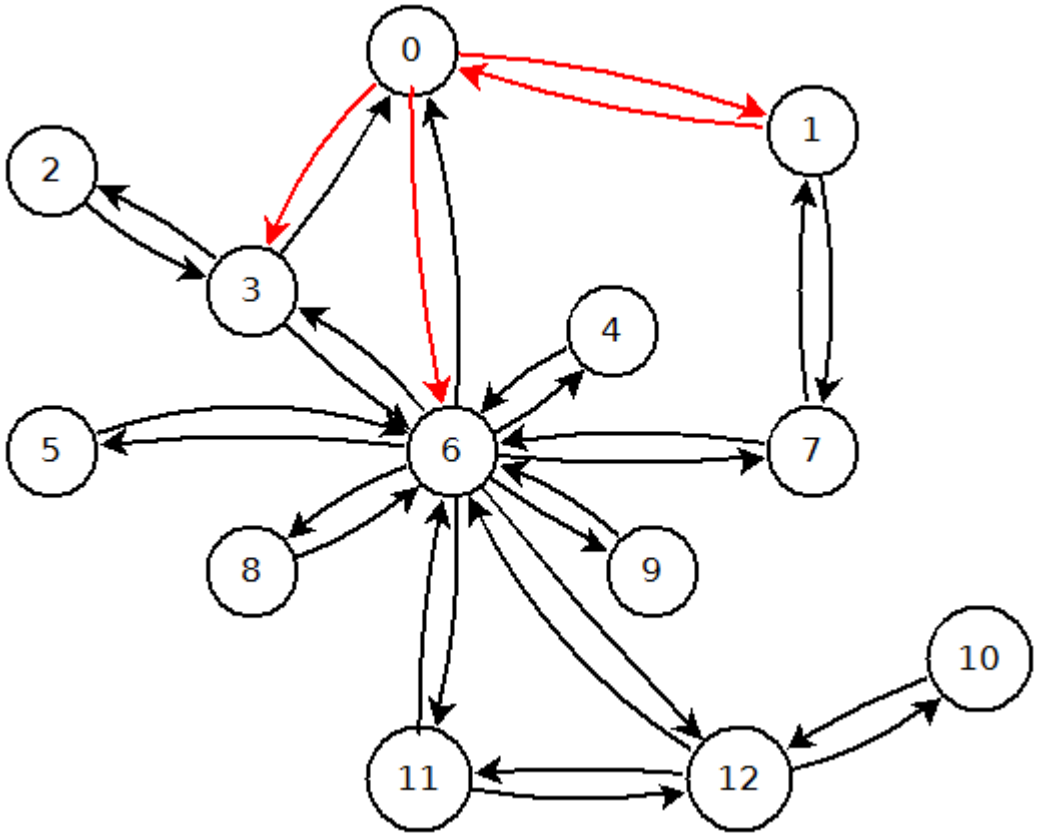
Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

0 1 2 3 4 5 6 7 8 9 10 11 12



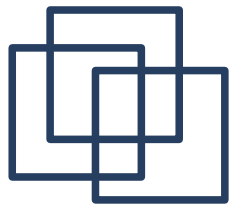
Распределение нагрузки по потокам



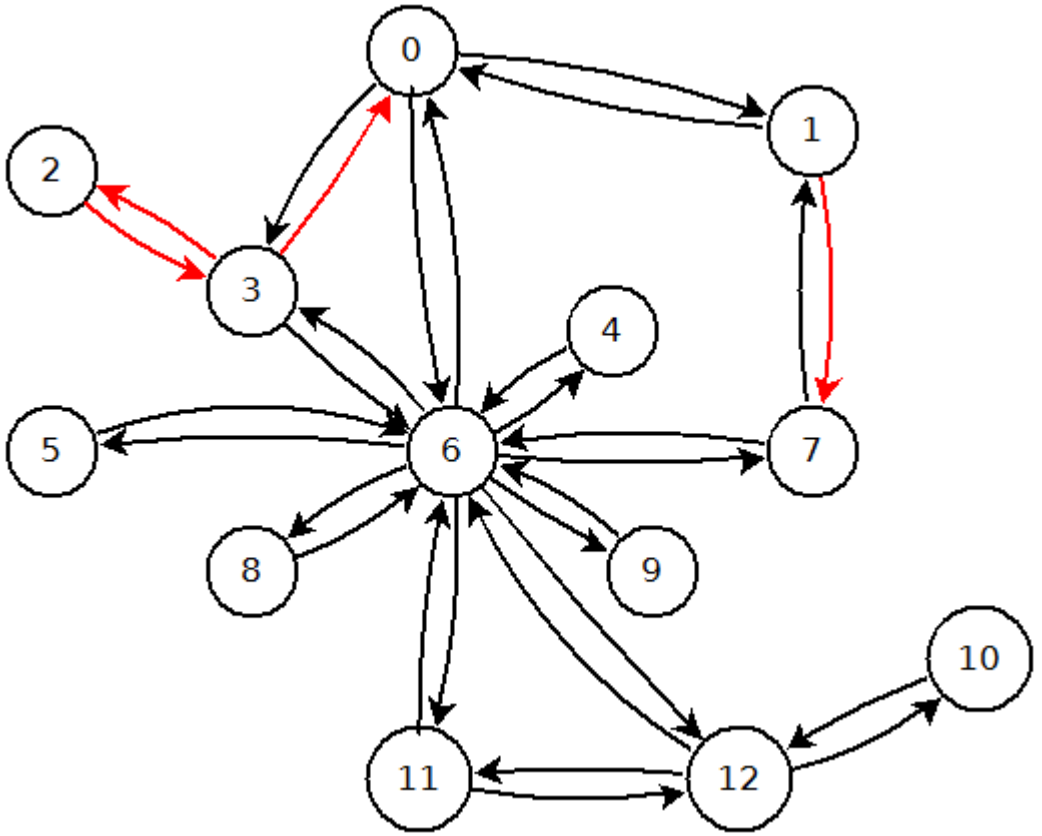
Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

Thread 0 → Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



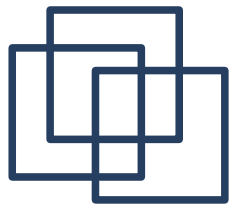
Распределение нагрузки по потокам



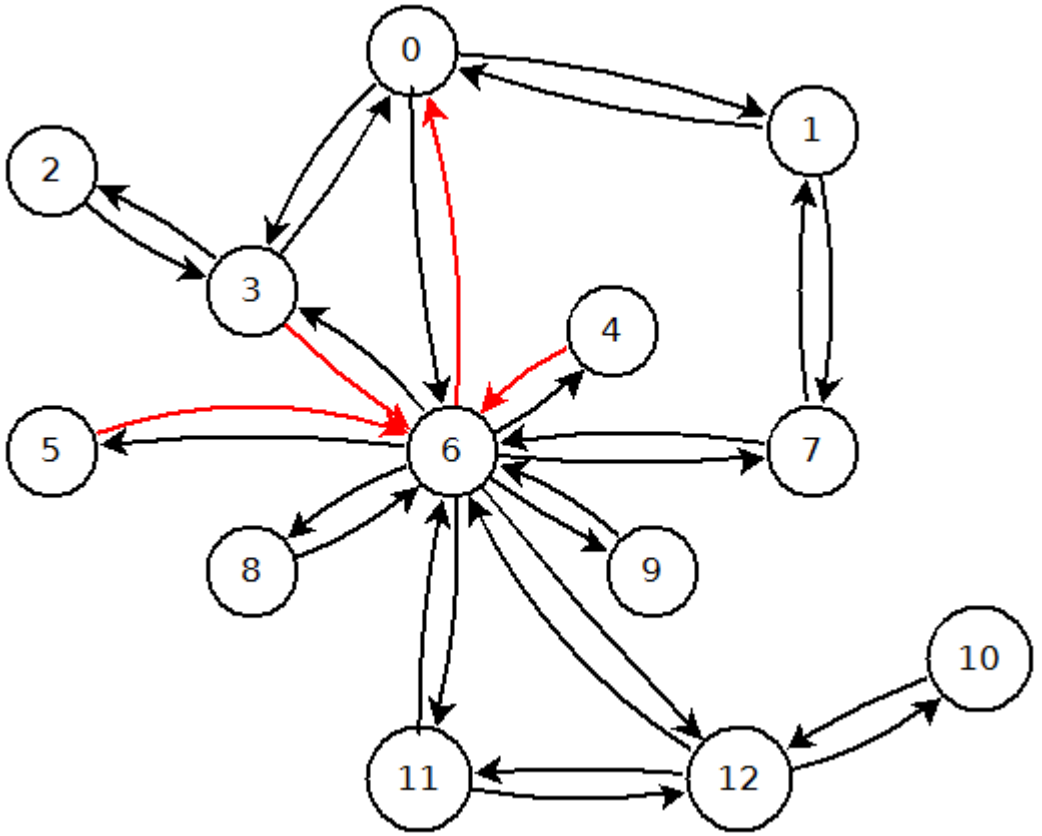
Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

Thread 1 → Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



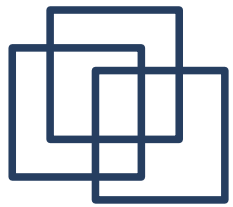
Распределение нагрузки по потокам



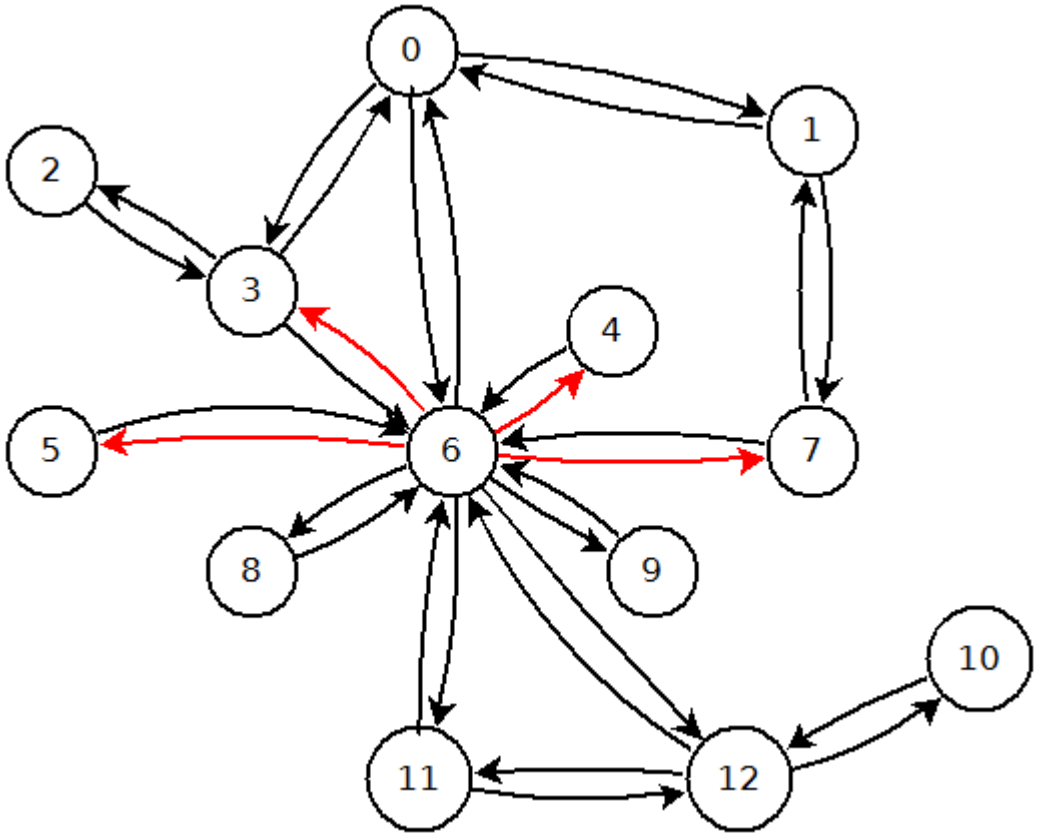
Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

Thread 2 → Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



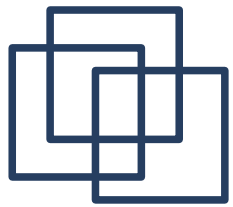
Распределение нагрузки по потокам



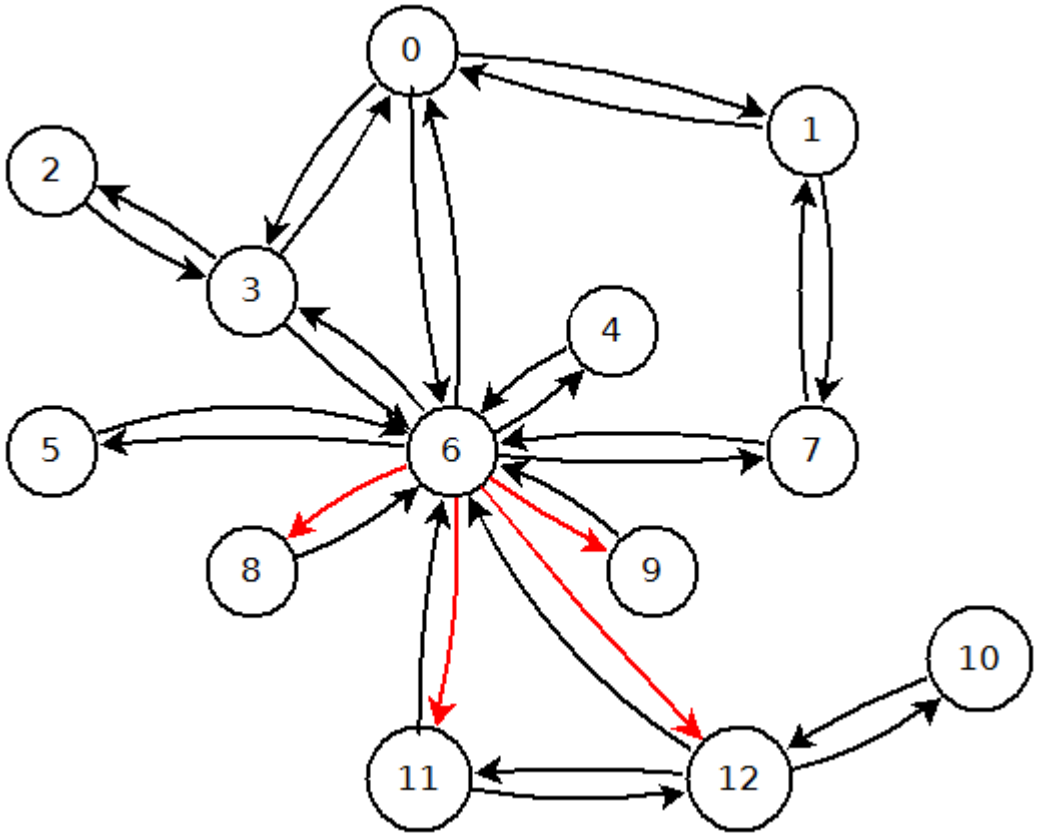
Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

Thread 3 → Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



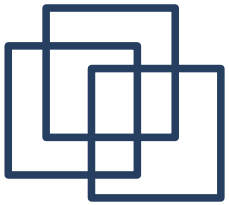
Распределение нагрузки по потокам



Vert: 0, 3, 5, 6, 9, 10, 11, 20, 22, 23, 24, 25, 27, 30

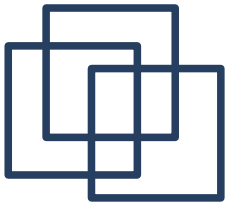
Edge: 1, 3, 6 | 0, 7 | 3 | 0, 2, 6 | 6 | 6 | 0, 3, 4, 5, 7, 8, 9, 11, 12 | 1, 6 | 6 | 6 | 12 | 6, 12 | 6, 10, 11

Thread 4 → Start_Vert: 0, 1, 3, 6, 6, 7, 10, 12



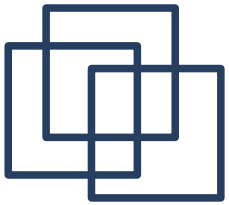
Распределение нагрузки по потокам

- Каждый поток выполняет фиксированный объем работы
  - Распределение работы при обработке «сложных» вершин
  - Объединение обработки «простых» вершин
- Оптимальная гранулярность для GPU Nvidia Tesla M2050/M2090: 128 ребер
- На каждой итерации происходит полный проход всех вершин и ребер
  - Каждый GPU должен обладать информацией обо всех вершинах



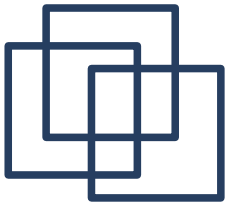
Синхронизация данных

- Информация о посещенных вершинах на всех GPU должна быть одинаковой
  - Синхронизация после каждой разметки вершин
- Способы
  - Коммуникации «точка-точка»
    - Возрастает сложность разметки
  - Коллективные операции
    - Увеличивается объем пересылаемых данных
- Решение
  - Использование битовой маски

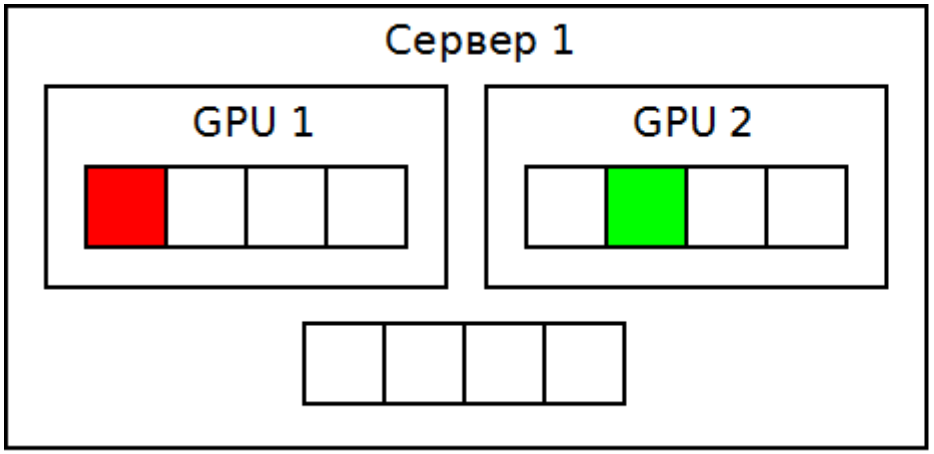


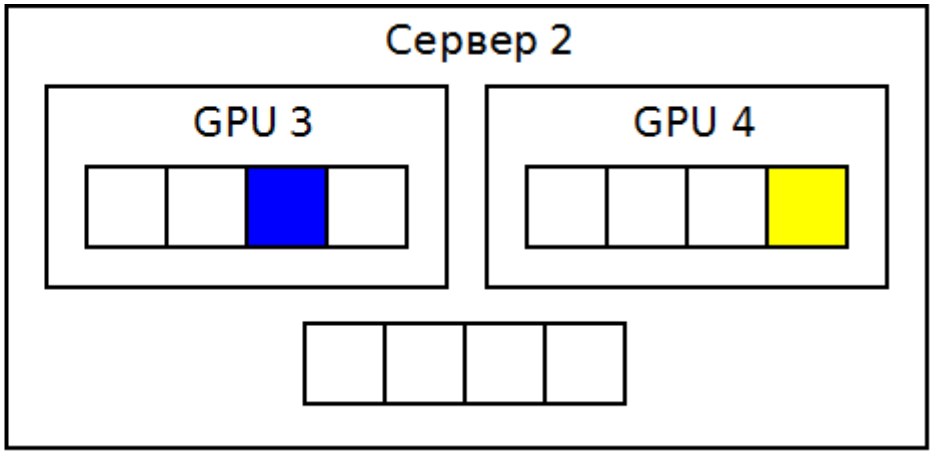
Битовая маска

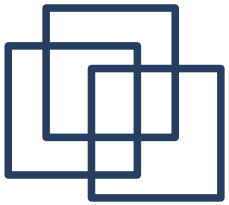
- Массив из беззнаковых целых чисел, использующийся как совокупность битов
  - 1 — вершина на текущей итерации
  - 0 — вершина не помечена
- Легкость транспортировки между узлами
  - Scale = 30
    - Граф целиком: 256 ГБ
    - Битовая маска: 128 МБ



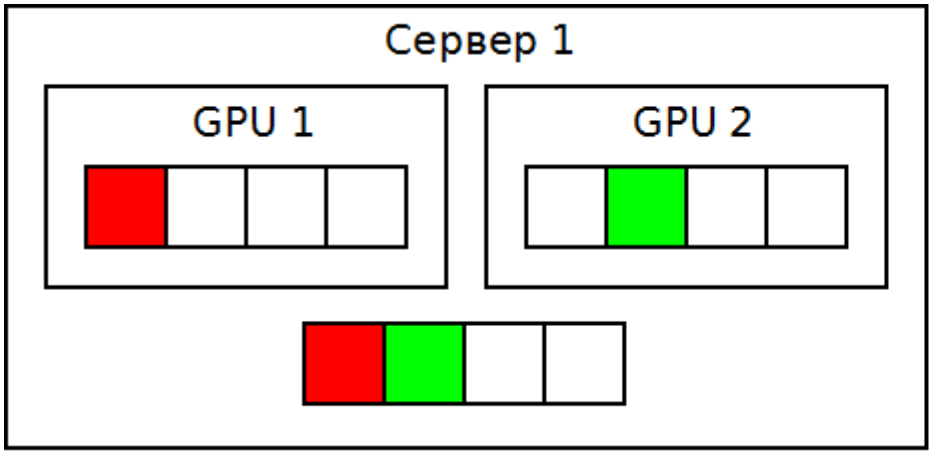
Синхронизация данных

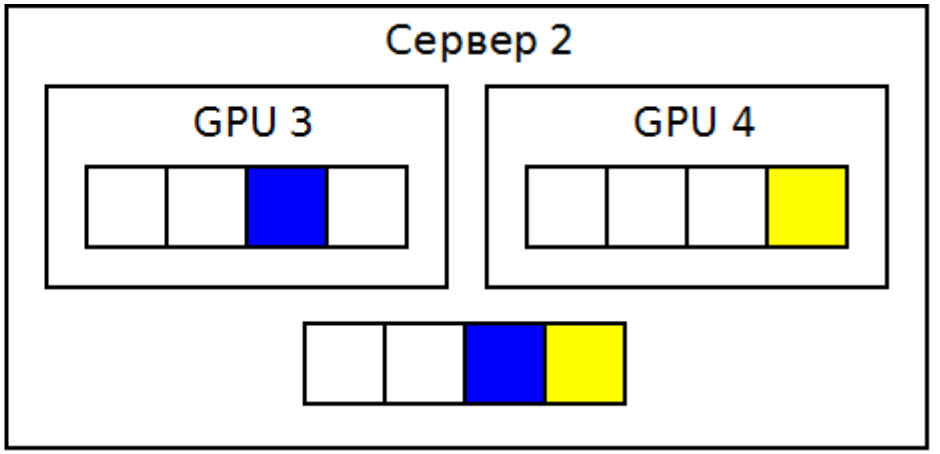


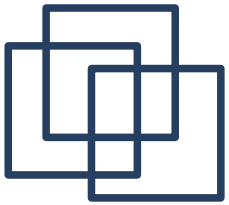




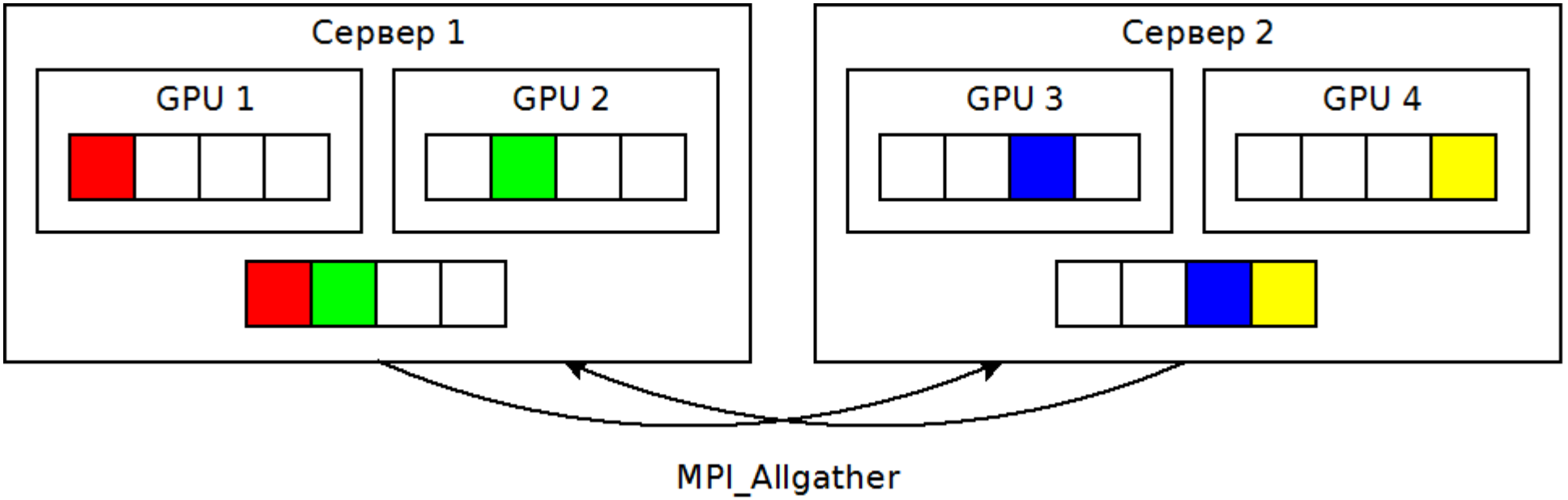
Синхронизация данных

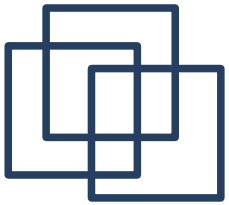




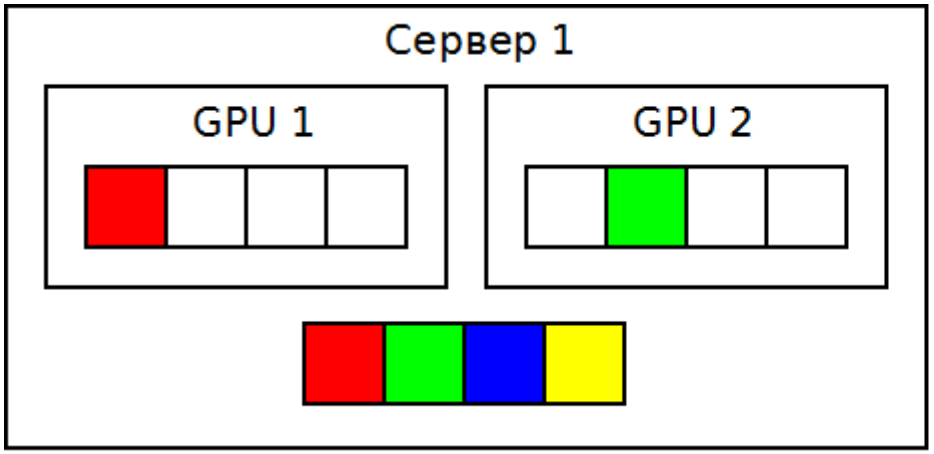


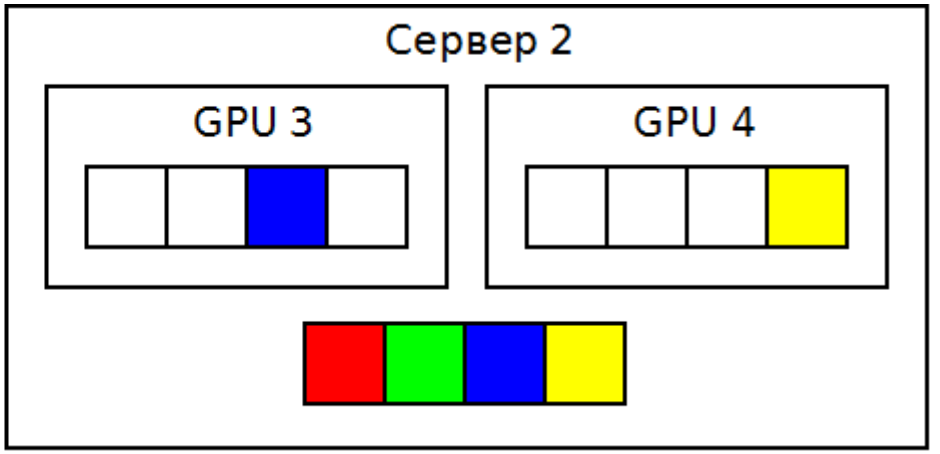
Синхронизация данных

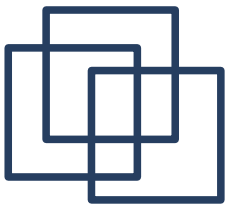




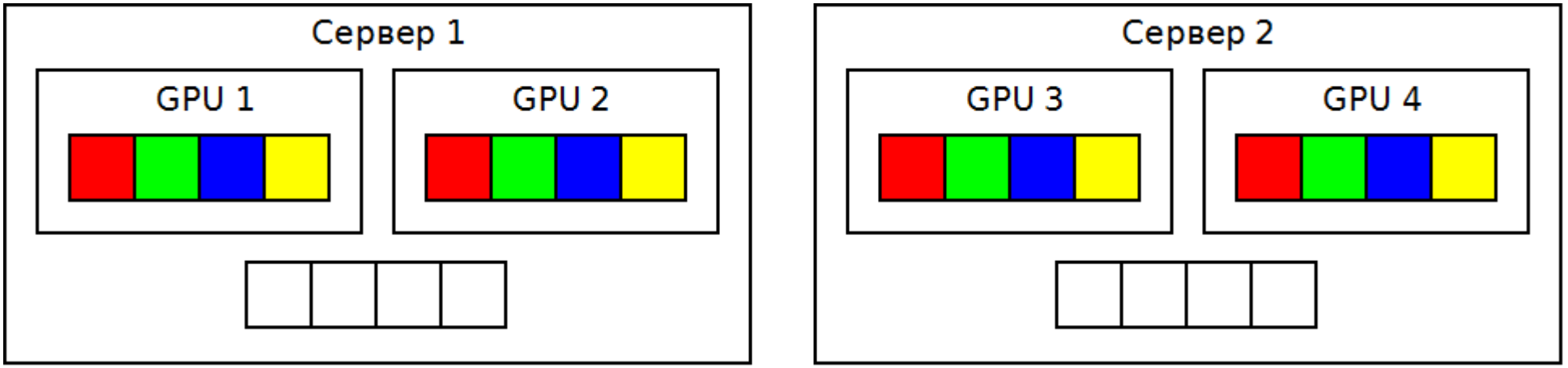
Синхронизация данных

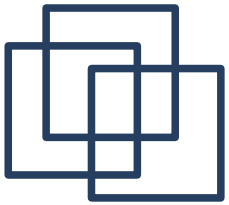






Синхронизация данных

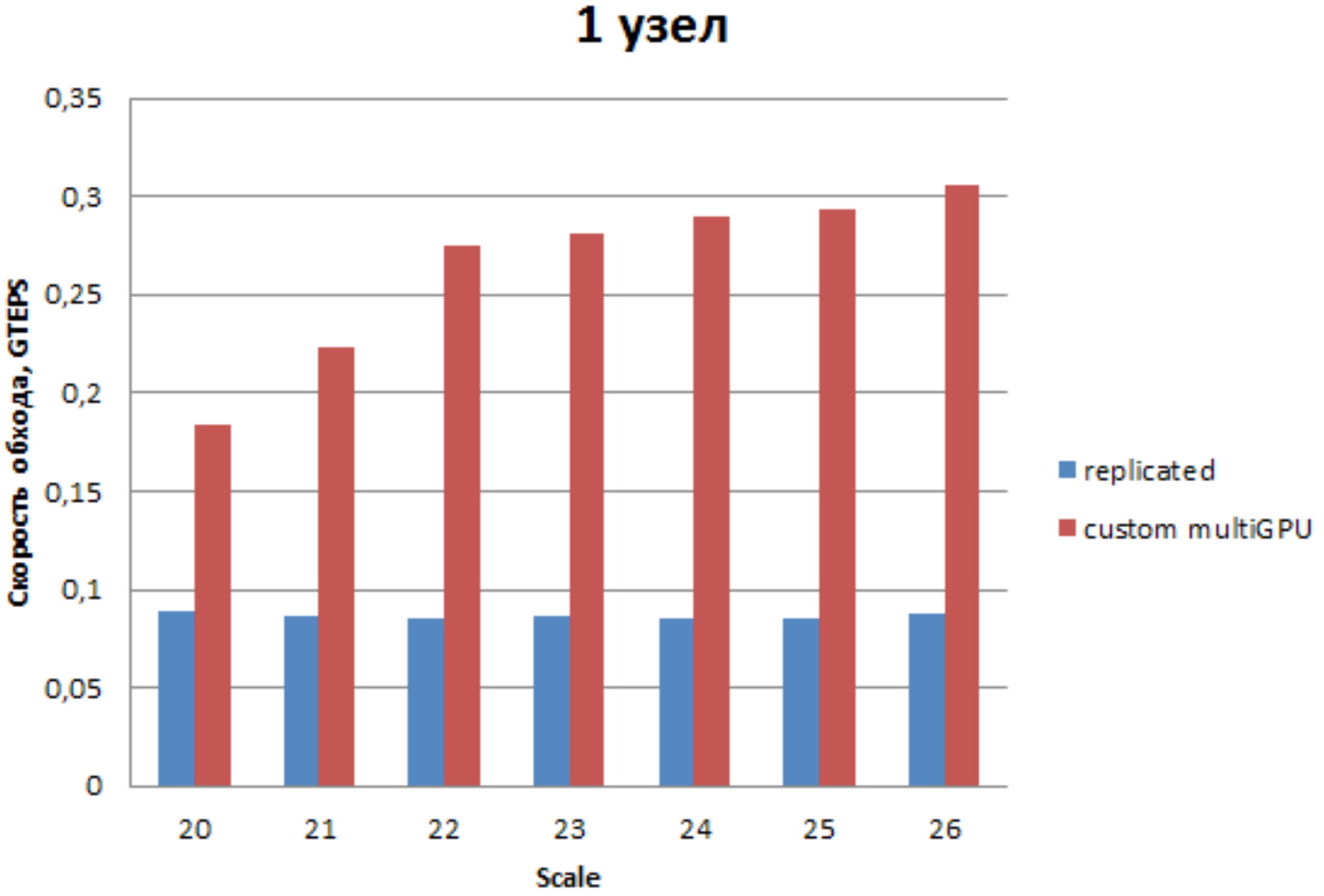


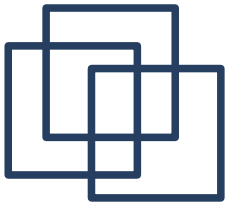


Тестирование

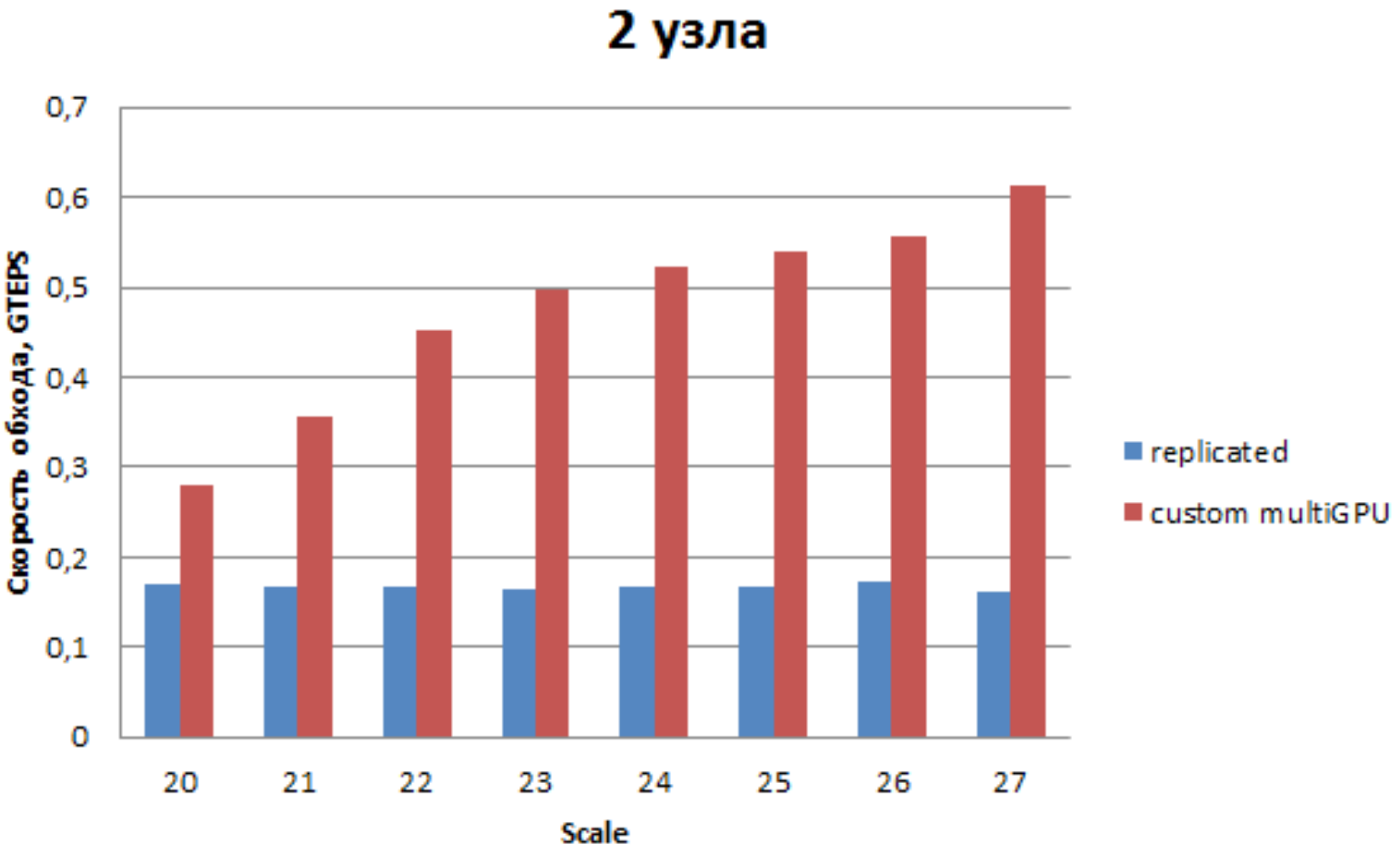
- Аппаратное обеспечение
  - Серверы HP ProLiant SL390s G7
    - 2 CPU Intel Xeon X5675
    - 8 GPU Nvidia Tesla M2050
  - IB 20 Гб/с
- Параметры
  - Scale от 20 до 30
  - Edgefactor = 16
  - Количество узлов от 1 до 16
- Версии теста Graph500
  - replicated
  - custom multiGPU

Результаты

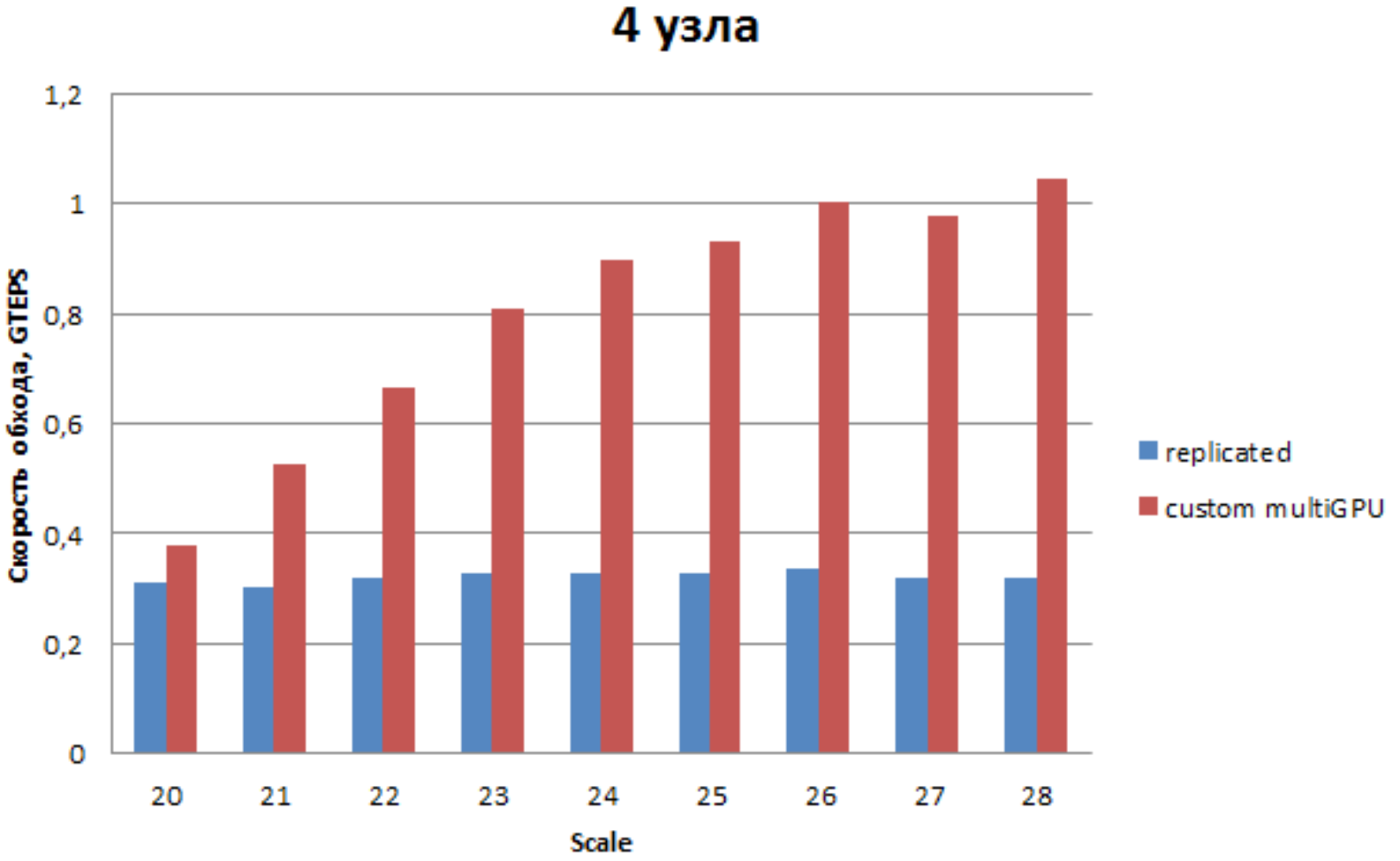

1 узел
Скорость обхода, GTEPS
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
20
21
22
23
24
25
26
Scale
replicated
custom multiGPU



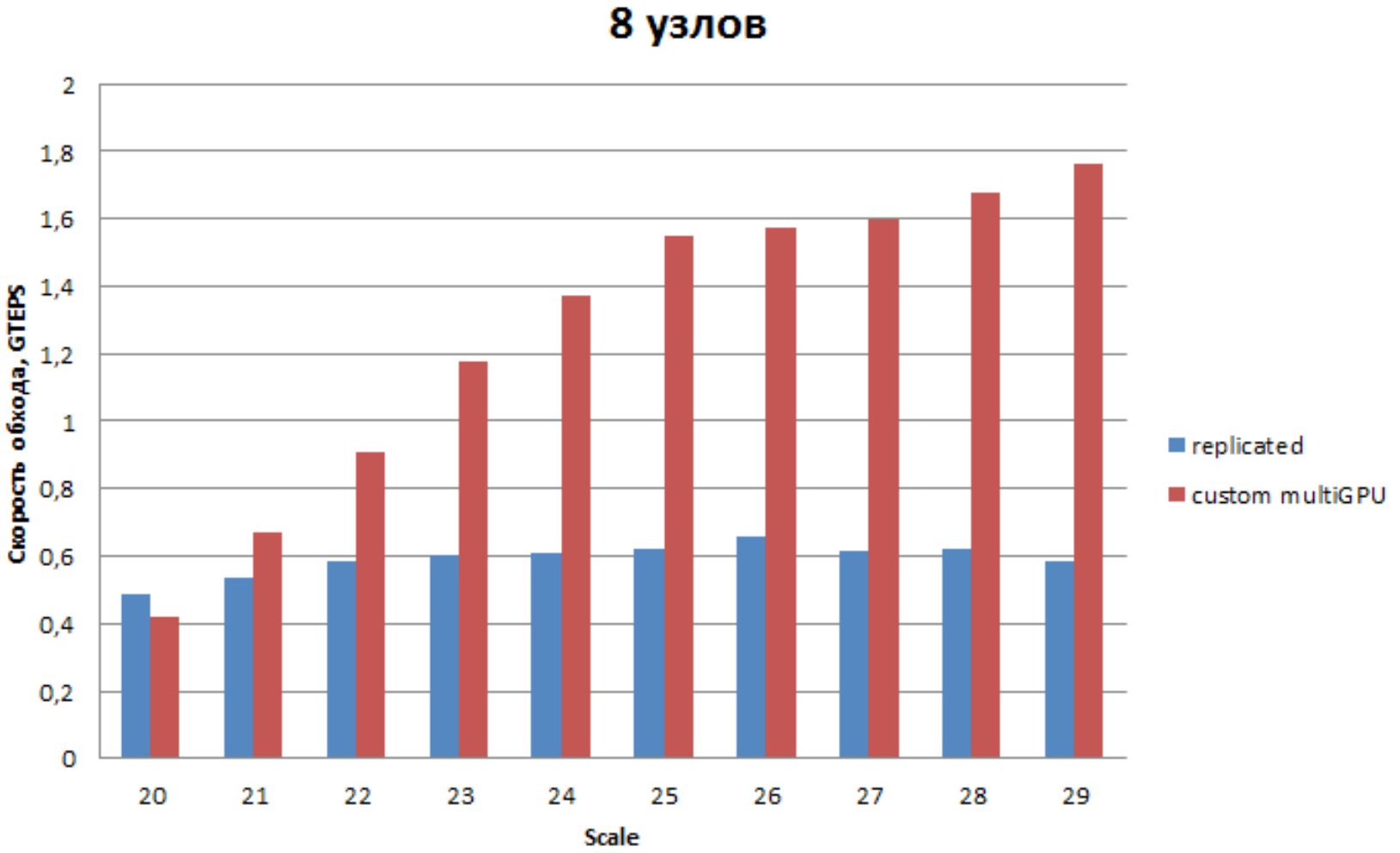
Результаты


2 узла
Скорость обхода, GTEPS
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
20
21
22
23
24
25
26
27
Scale
replicated
custom multiGPU

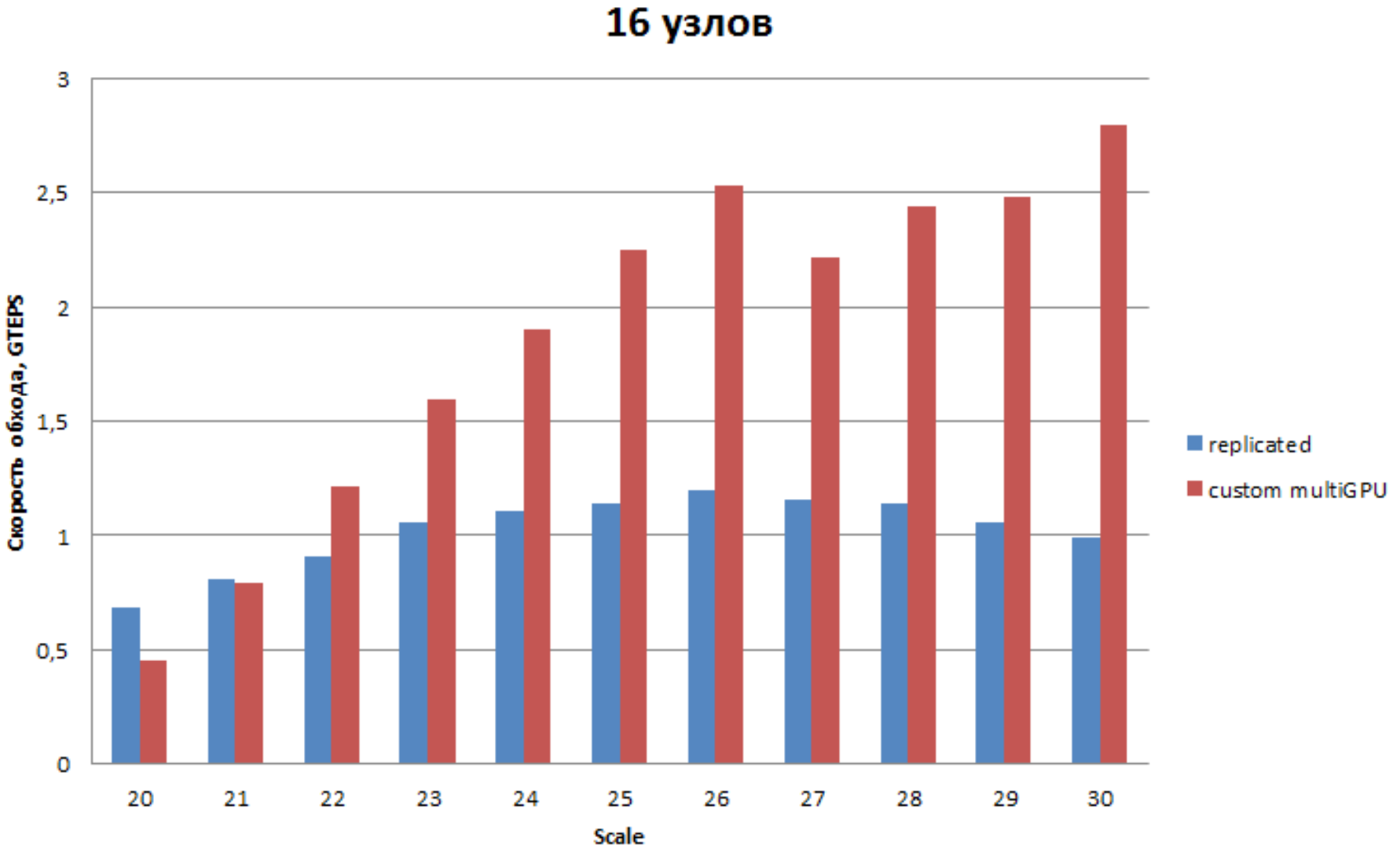
Результаты

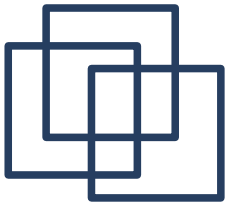

4 узла
Скорость обхода, GTEPS
1,2
1
0,8
0,6
0,4
0,2
0
20
21
22
23
24
25
26
27
28
Scale
replicated
custom multiGPU

Результаты


8 узлов
Скорость обхода, GTEPS
2
1,8
1,6
1,4
1,2
1
0,8
0,6
0,4
0,2
0
20
21
22
23
24
25
26
27
28
29
Scale
replicated
custom multiGPU

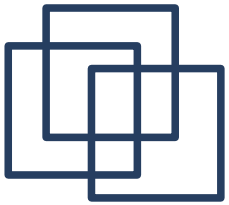
Результаты


16 узлов
Скорость обхода, GTEPS
3
2,5
2
1,5
1
0,5
0
20
21
22
23
24
25
26
27
28
29
30
Scale
replicated
custom multiGPU



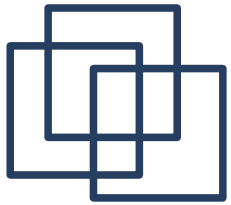
Сильное и слабое масштабирование

- Сильное масштабирование
  - «Strong Scaling»
  - Характеризует способность алгоритма к распараллеливанию
- Слабое масштабирование
  - «Weak Scaling»
  - Характеризует влияние интерконнекта на масштабируемость

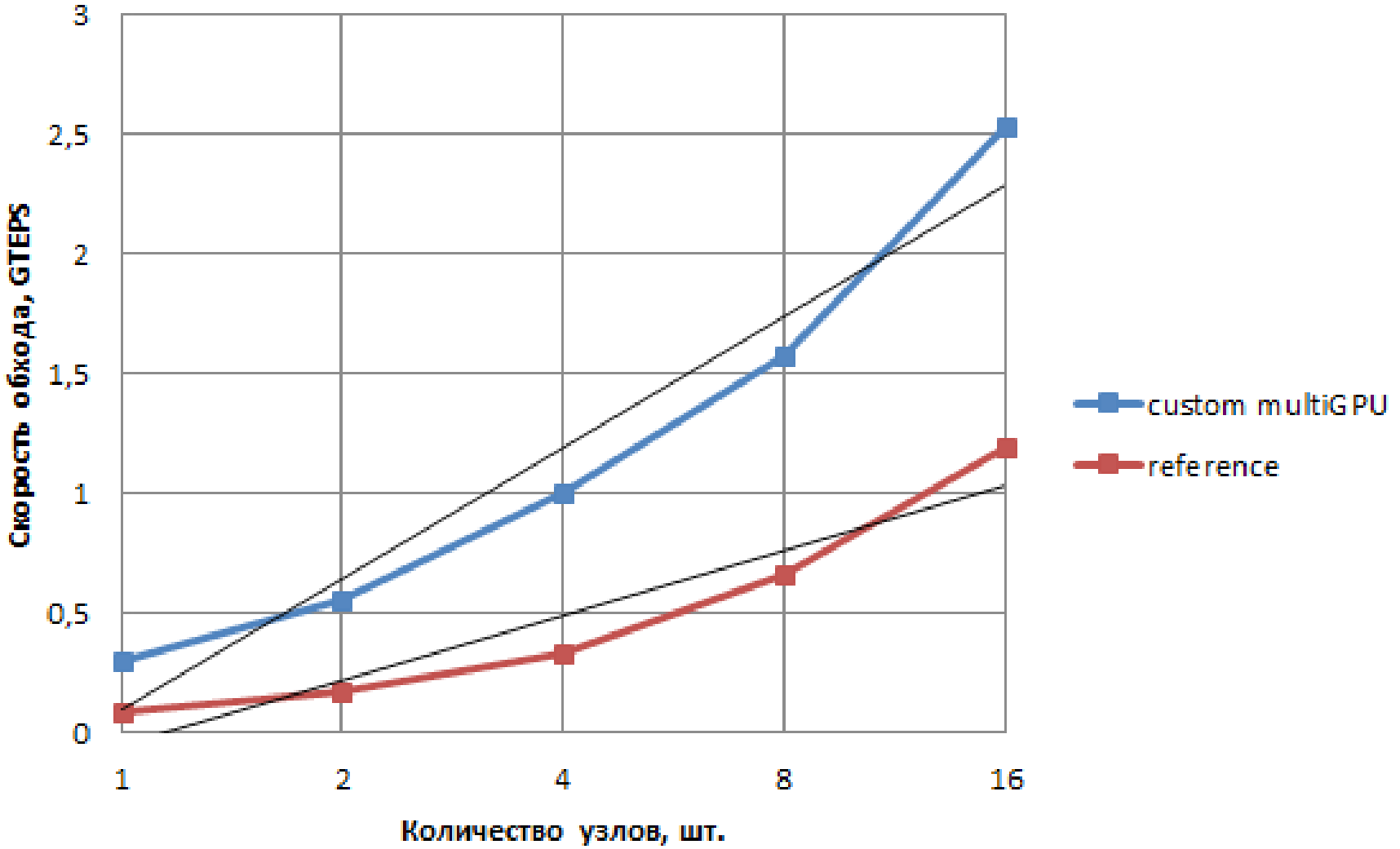


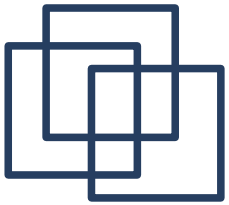
Сильное масштабирование

- Фиксируем размер задачи
  - Scale = 26
- Варьируем количество узлов
  - 1, 2, 4, 8, 16



Сильное масштабирование

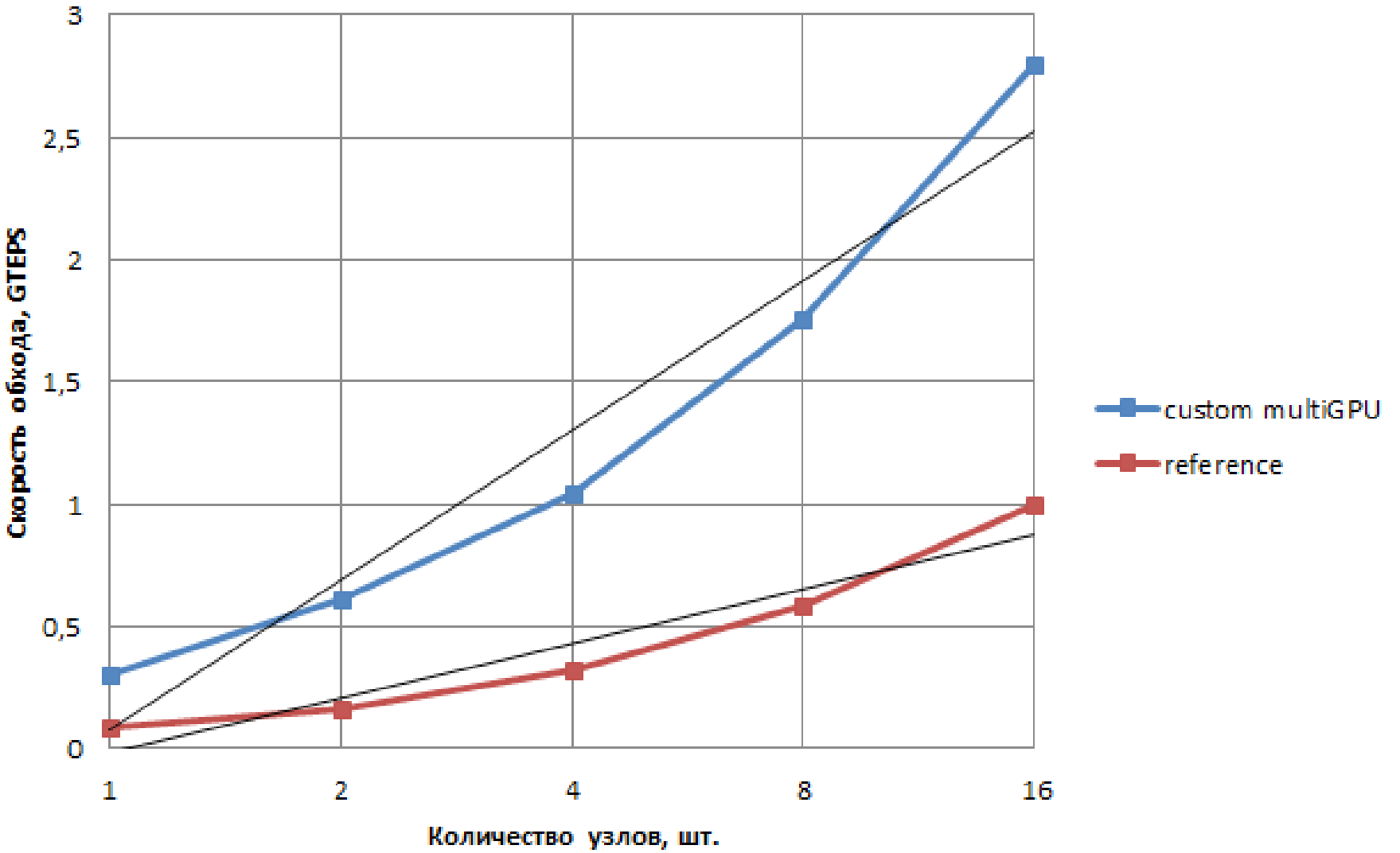


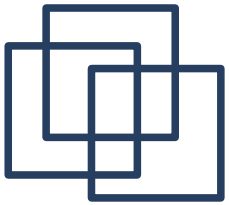


Слабое масштабирование

- Фиксируем размер задачи, приходящейся на один узел
  - 1 узел: Scale = 26
  - 2 узла: Scale = 27
  - 4 узла: Scale = 28
  - 8 узлов: Scale = 29
  - 16 узлов: Scale = 30

Слабое масштабирование

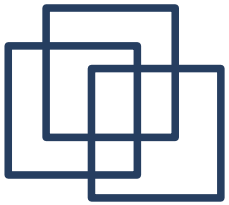




Сравнение времени выполнения

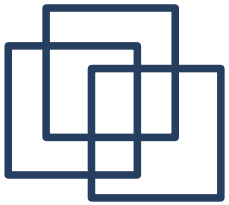
- Scale = 30
- 16 узлов

	reference	**custom multiGPU**
Копирование данных, с	Отсутствует	2.5
Подготовка к запуску поиска в ширину, с	Отсутствует	0.2
Поиск в ширину, с	17.3	3.4
Общее время, с	**17.3**	**6.1**



Другая multiGPU-реализация

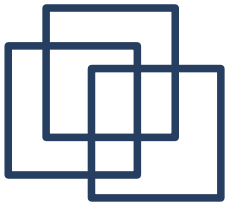
- Разработчики
  - Enrico Mastrostefano
  - Massimo Bernaschi
- Организация
  - http://on-demand.gputechconf.com/gtc/2012/



Сравнение

- Scale = 25

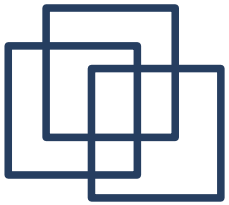
	ИММ	**CSCS**
Модель GPU	M2050	M2070
IB	DDR	QDR
Количество GPU	64	64
Количество GPU в узле	8	2
Копирование данных	+	н.д.
Выходные данные	Массивы pred и dist	Массив pred
Скорость обхода, GTEPS	**1.55**	**1.25**



Сравнение

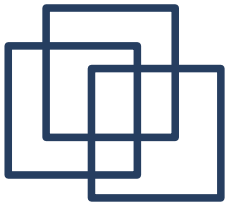
- Scale = 28

	ИММ	**CSCS**
Модель GPU	M2050	M2090
IB	DDR	QDR
Количество GPU	128	128
Количество GPU в узле	8	2
Копирование данных	+	н.д.
Выходные данные	Массивы pred и dist	Массив pred
Скорость обхода, GTEPS	**2.79**	**3.06**



Выводы

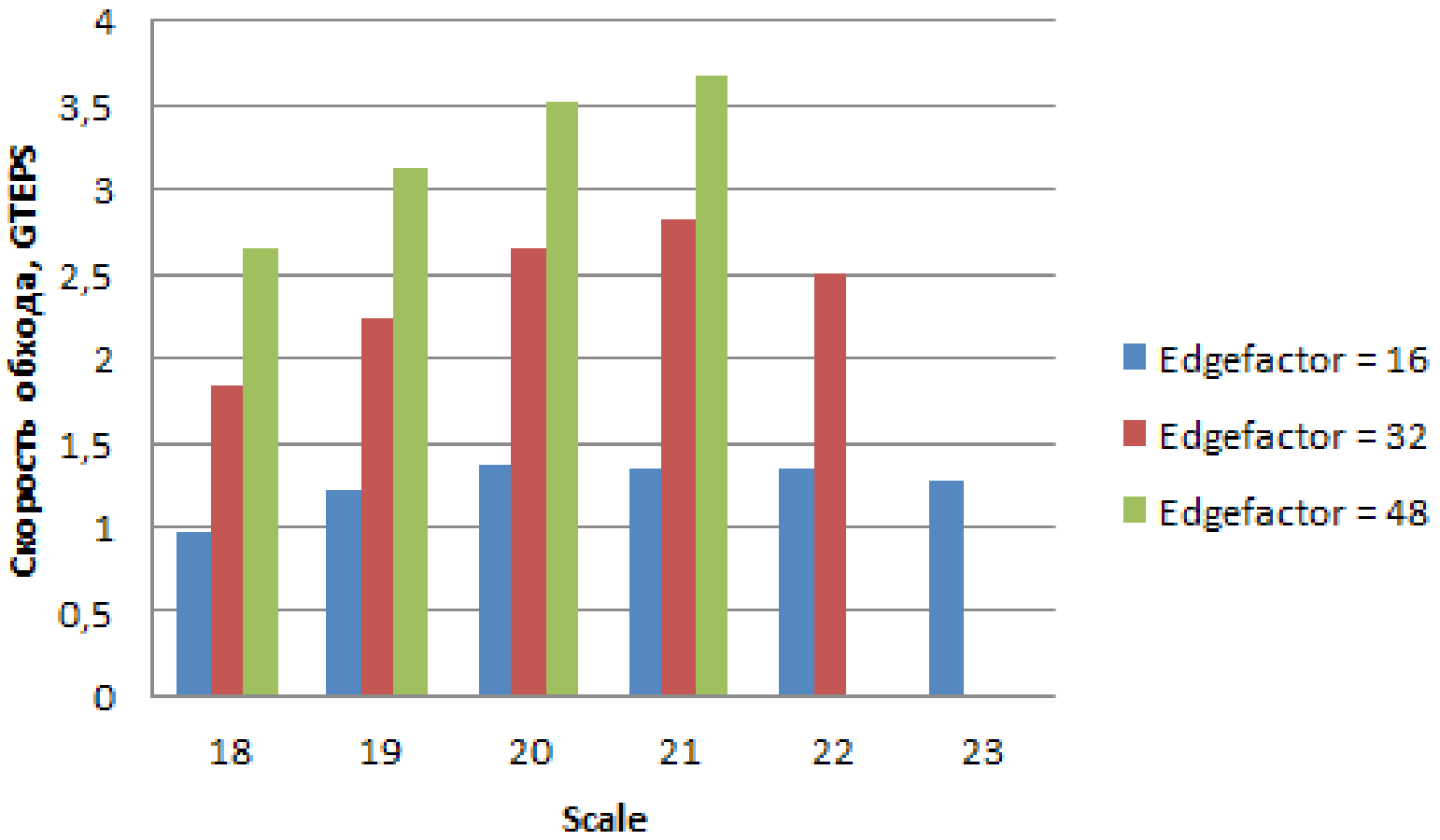
- Разработан метод балансировки нагрузки, позволяющий равномерно загружать большое количество вычислительных потоков
- Метод использован при создании собственной реализации теста производительности Graph500
  - Достигнуто ускорение более чем в два раза по сравнению с replicated-реализацией

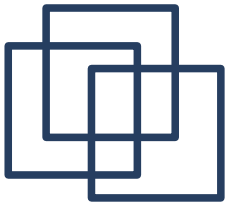


P.S.

- Поиск в ширину на одном GPU
  - Начальные итерации: top-down
  - Заключительные итерации: bottom-up
- Балансировка нагрузки
- Разметка массива уровней и массива родителей

P.S.


Скорость обхода, GTEPS
4
3,5
3
2,5
2
1,5
1
0,5
0
18
19
20
21
22
23
Scale
Edgefactor = 16
Edgefactor = 32
Edgefactor = 48



Вопросы?