Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Параллельный алгоритм построения остовного дерева для больших графов на мультипроцессоре

Зайцев Вадим, магистрант

Научный руководитель:
Калгин Константин Викторович, к.ф.-м.н., ИВМиМГ

Постановка задачи

- **Остовное дерево** - ациклический связный подграф данного связного неориентированного графа, в который входят все его вершины.
- **Минимальное остовное дерево** - остовное дерево с минимальной суммой весов рёбер.

- **Дано**: взвешенный неориентированный связный граф G(V, E).
- **Задача**: построить минимальное остовное дерево.

Алгоритмы построения MST

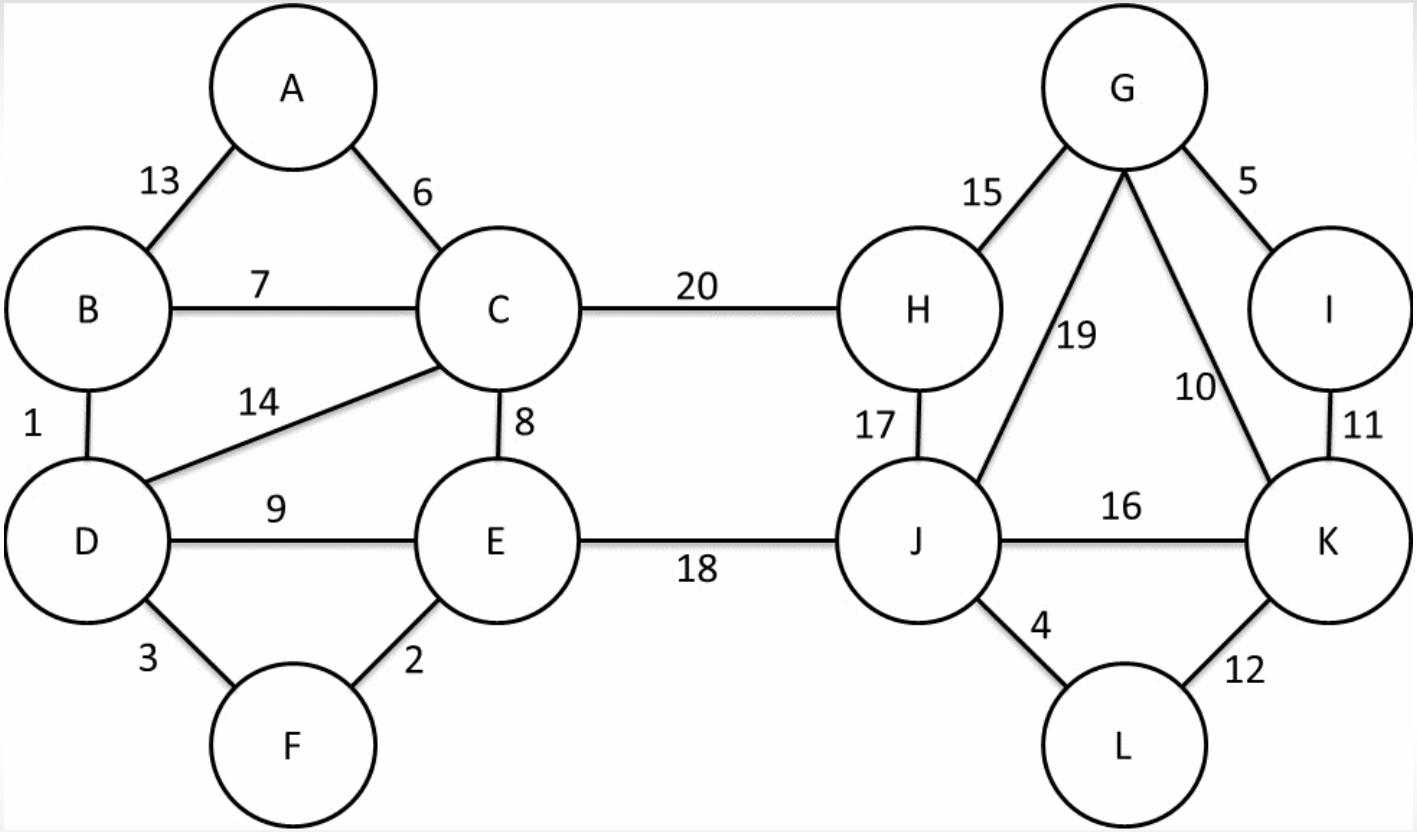
- **Алгоритм Примы**
  - Выбираем произвольную вершину графа и начинаем растить минимальное дерево.
  - Выбираем минимальное ребро, исходящее из уже построенного дерева, и добавляем новую вершину к дереву.
- **Алгоритм Крускала**
  - Сортируем рёбра по неубыванию веса.
  - По очереди перебираем рёбра и добавляем в остов те, которые не создают цикла.
- **Алгоритм Борувки**
  - На каждой итерации для каждой компоненты находим минимальное ребро и объединяем компоненты по найденным рёбрам

Алгоритм решения (Борувки)

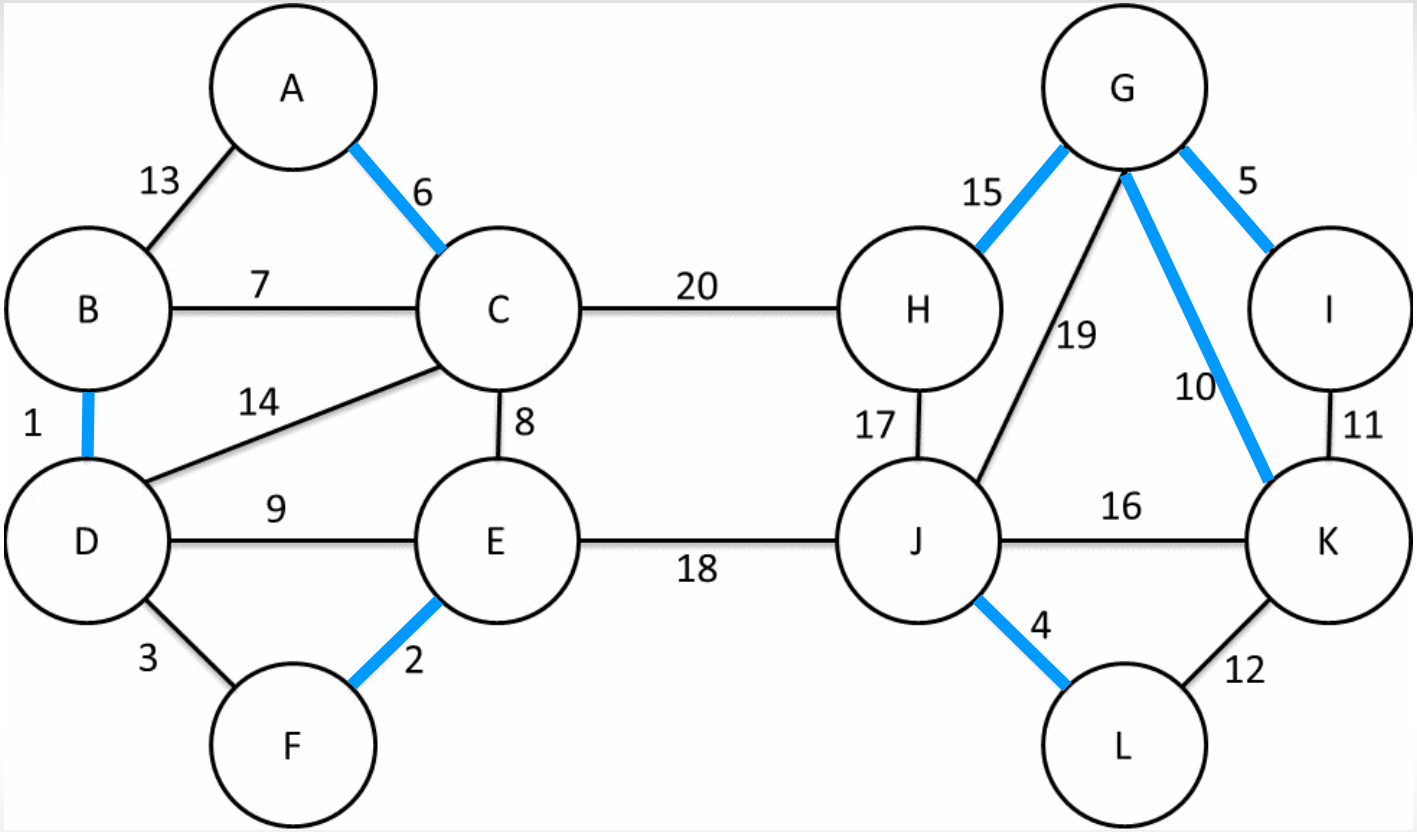
1. Дан граф G(V, E).
2. Изначально полагаем T - пустым множеством рёбер. В дальнейшем работает с графами G(V, E) и G(V, T).
3. Далее, пока T не является деревом:
   a) Для каждой компоненты в G(V, T) найдём минимальное ребро из E, соединяющее его с другой компонентой
   b) Добавим все найденные рёбра в T

Временная сложность: O(E log(V))

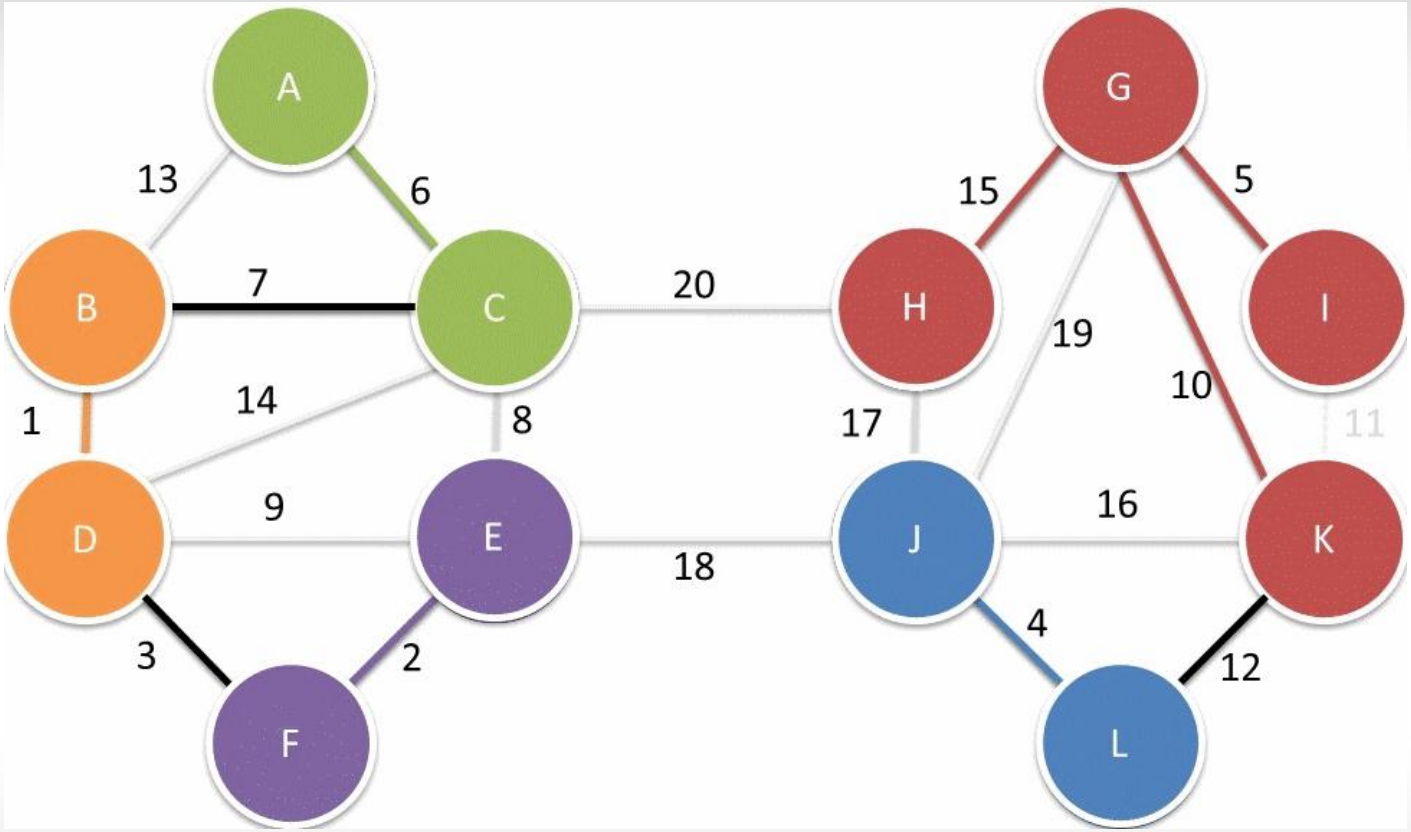
Алгоритм Борувки



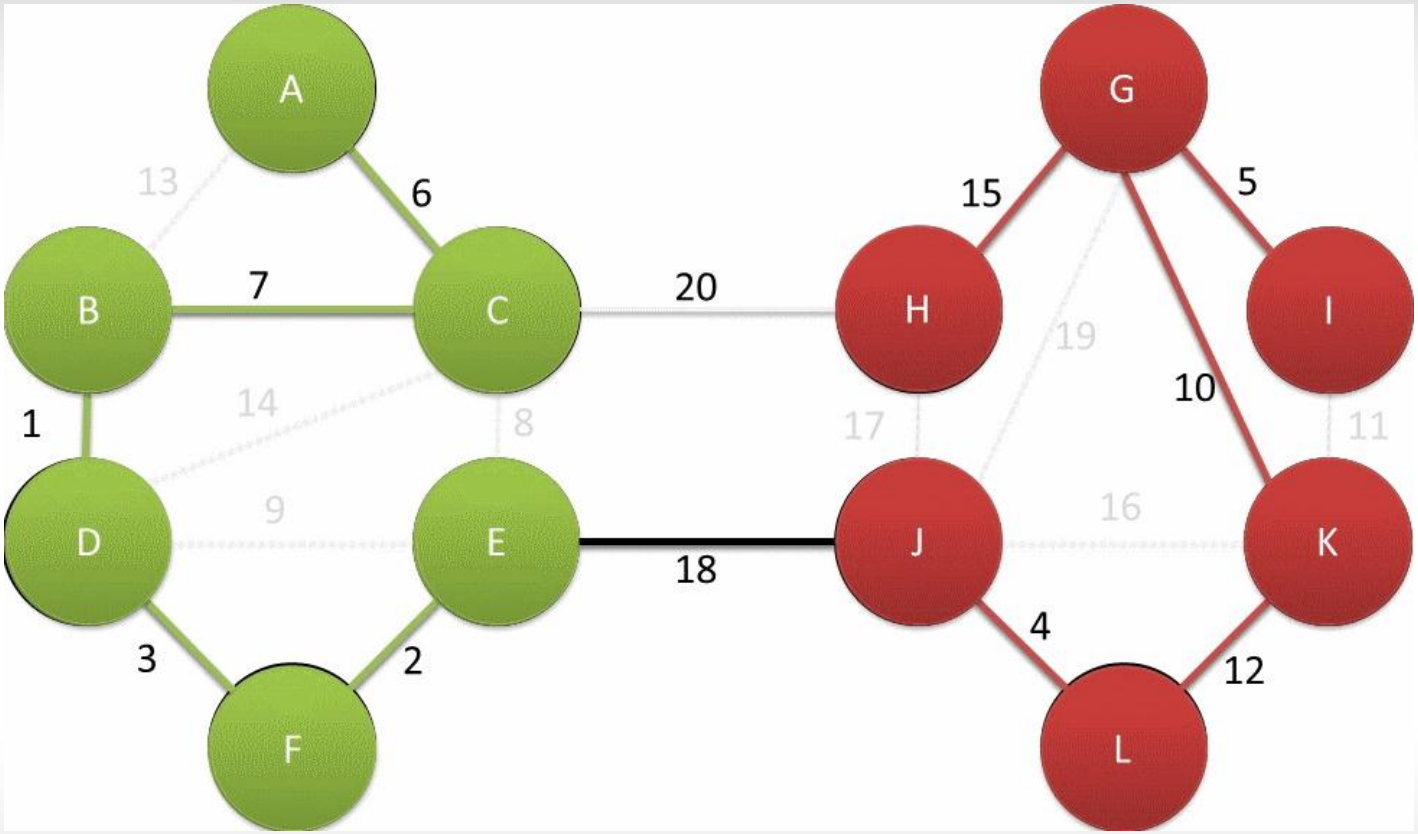
Алгоритм Борувки



Алгоритм Борувки



Алгоритм Борувки



Анализ алгоритмов

- Алгоритм Крускала и алгоритм Примы:
  - зависимость между шагами алгоритма
  - большое количество шагов
  - малая вычислительная сложность одного шага
- Алгоритм Борувки:
  - зависимость между шагами алгоритма
  - малое количество шагов
  - большая вычислительная сложность одного шага

Тестирование

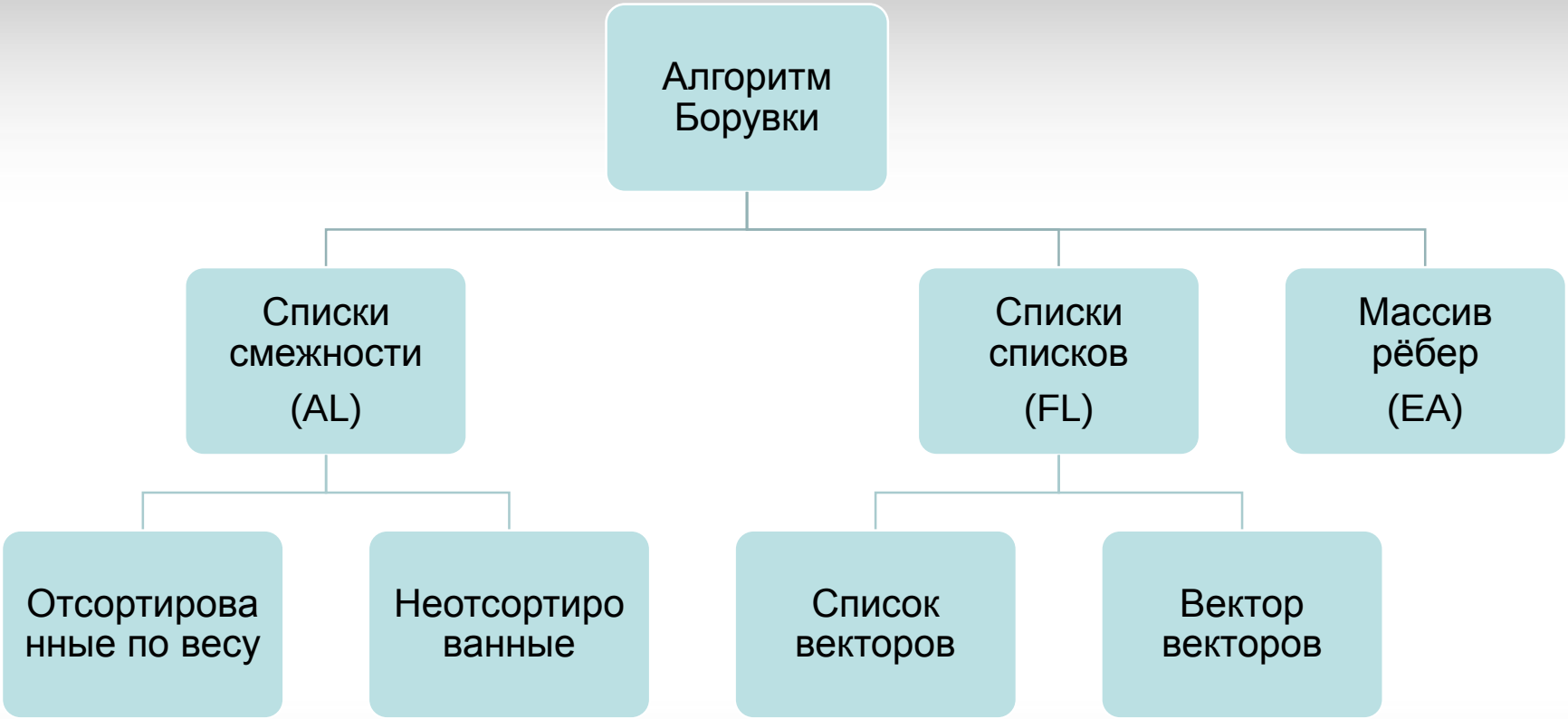
Тестирование проводилось на графе RMAT-22:

- 2^{22} вершин
- средняя степень вершины: 32
- 2^{27} рёбер
- Объём данных: 2Гб

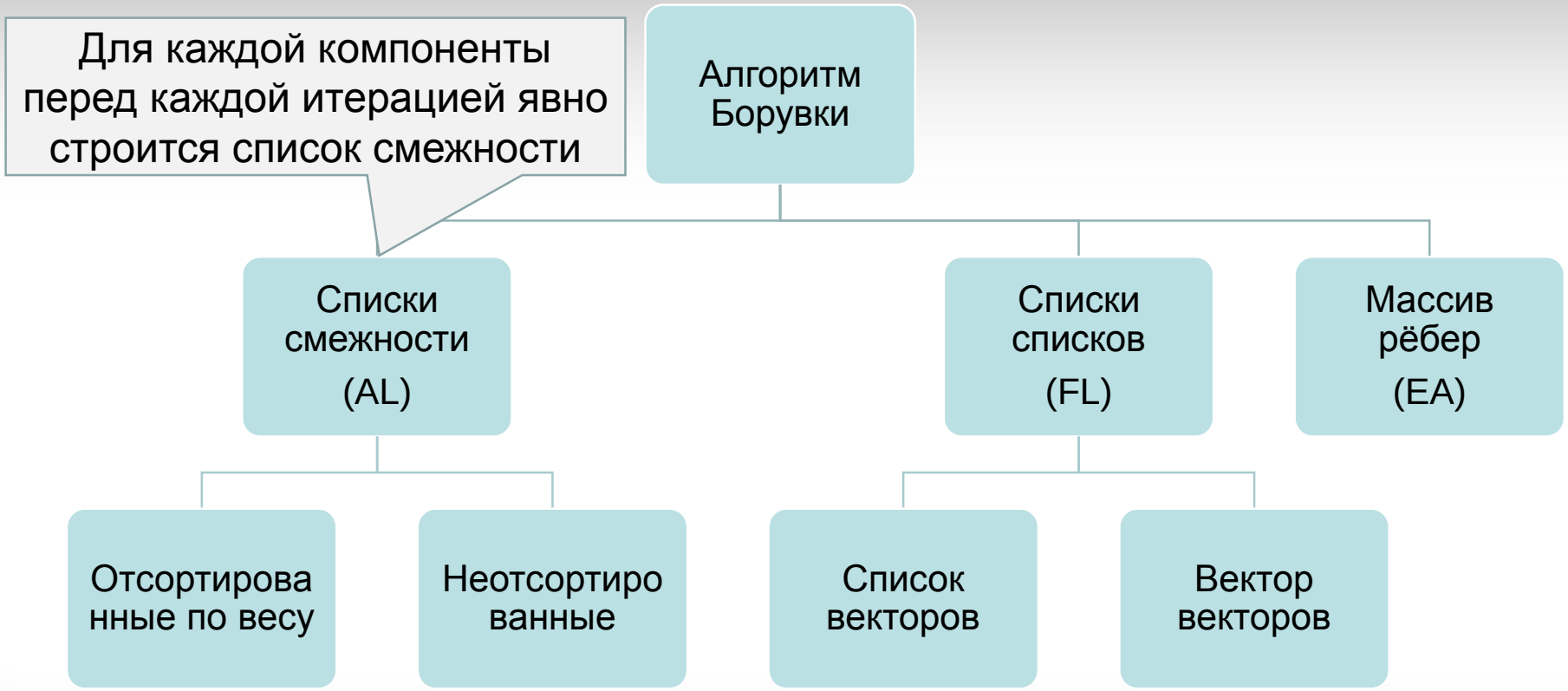
Входной граф хранится в формате CSR (compressed sparse row matrix)

Тестирование проводилось на узле МСЦ mvs1p1: 2 x Intel Xeon CPU E5-2690 (2 x 8 ядер 2.9 GHz, 32KB L1, 256KB L2, 20MB L3)

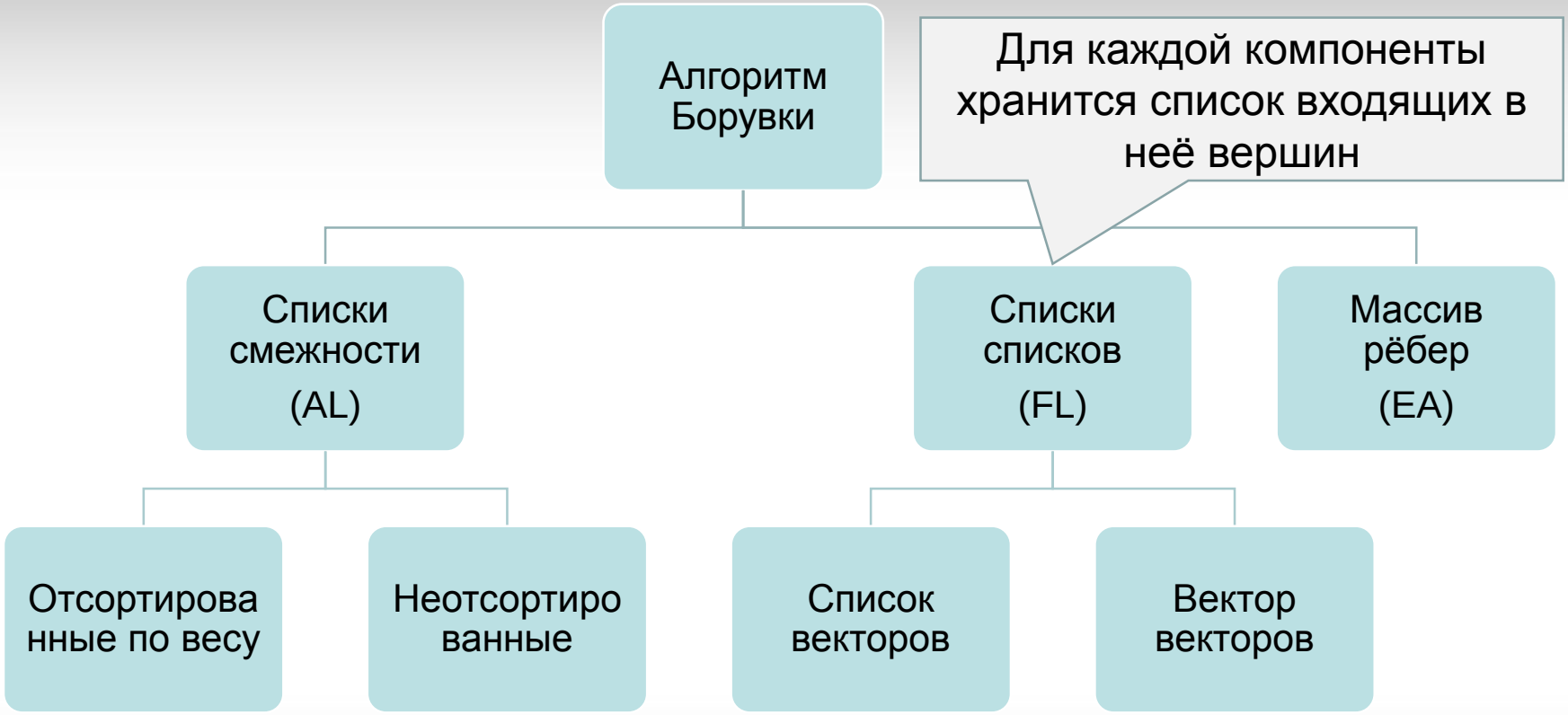
Подходы к хранению графа



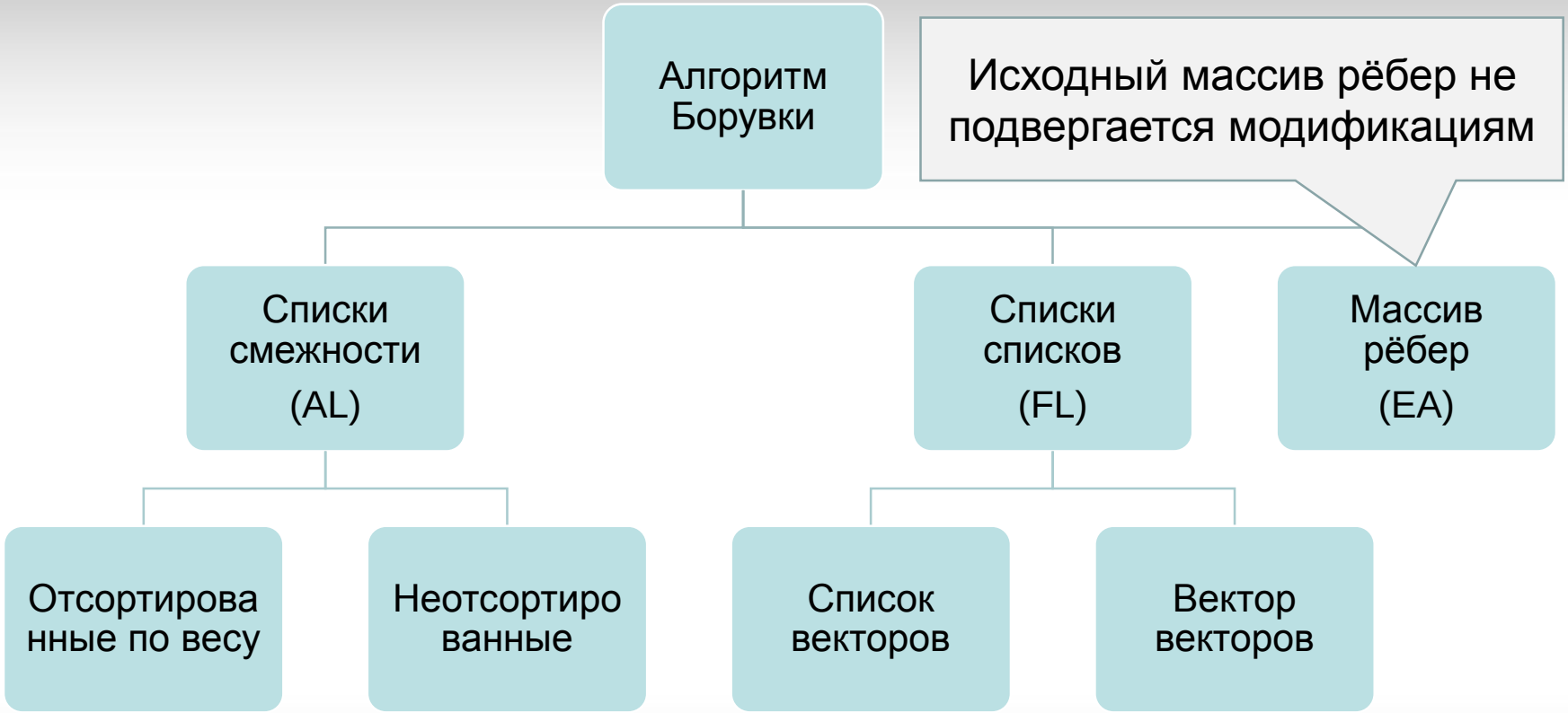
Подходы к хранению графа



Подходы к хранению графа



Подходы к хранению графа



Списки смежности: алгоритм

Для каждой **компоненты** хранится список всех инцидентных ей **рёбер**.

На каждой итерации алгоритма Борувки:

a. каждый поток для **каждой своей** компоненты находит инцидентное ей ребро минимального веса
b. объединение компонент по найденным рёбрам (BFS)
c. Pointer Jumping
d. слияние списков рёбер
   - обновление номера компоненты
   - удаление петель

Списки смежности: модификации

2 варианта слияния:

1. Отсортированные списки смежности
   - поиски минимального ребра за O(1)
   - более длительное слияние списков
2. Неотсортированные списки смежности
   - сложнее найти минимально ребро
   - слияния списков происходит простым копированием

Выявленные проблемы:

- Слияние списков рёбер занимает много времени
- Дисбаланс на последних итерациях

Сделанные оптимизации:

- Использование кучи при слиянии

Списки списков смежности: алгоритм

Для каждой **компоненты** хранится списков входящих в неё **вершин**.

На каждой итерации алгоритма Борувки:

a. **каждый поток для каждой своей компоненты находит инцидентное ей ребро минимального веса**
b. объединение компонент по найденным рёбрам (BFS или PJ)
c. Pointer Jumping
d. объединение списков **вершин**

Списки списков смежности: модификации

2 варианта хранения вершин, входящих в компоненту:

1. Односвязный список вершин
   - объединение двух компонент за O(1)
   - случайные обращения в память при обходе списка
2. Вектор (динамический массив)
   - объединение за время, пропорциональное длине короткого списка
   - нет случайных «прыжков» по памяти

Списки списков смежности: результаты

Выявленные проблемы:

- Объединять списки по-прежнему долго
- Дисбаланс на последних итерациях

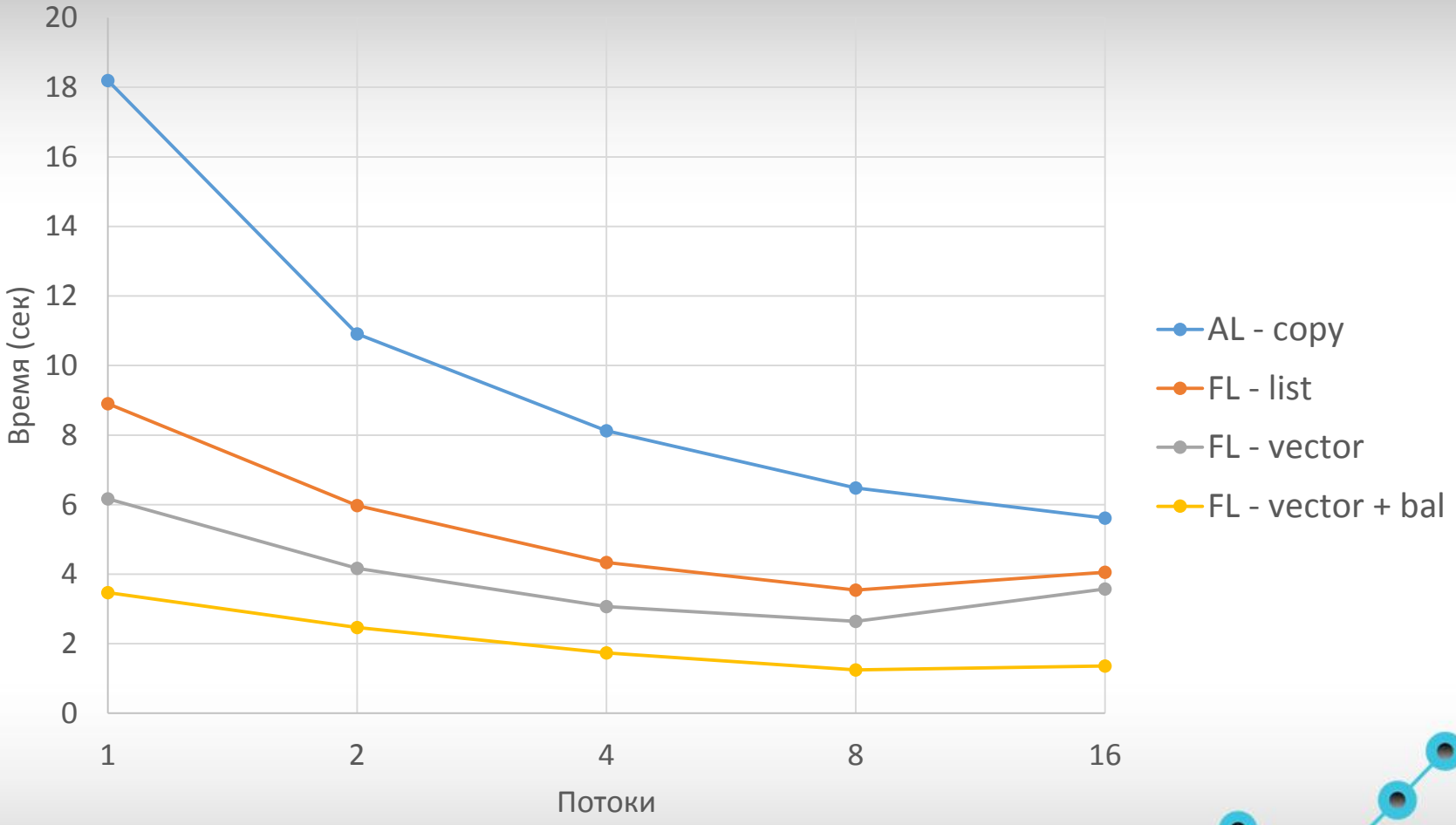
Сделанные оптимизации:

- Разделение шага поиска минимального ребра на два: отдельно для вершин с малой и высокой степенями

Возможные оптимизации:

- Объединять вершины в блоки для ускорения копирования

Списки списков смежности: результаты



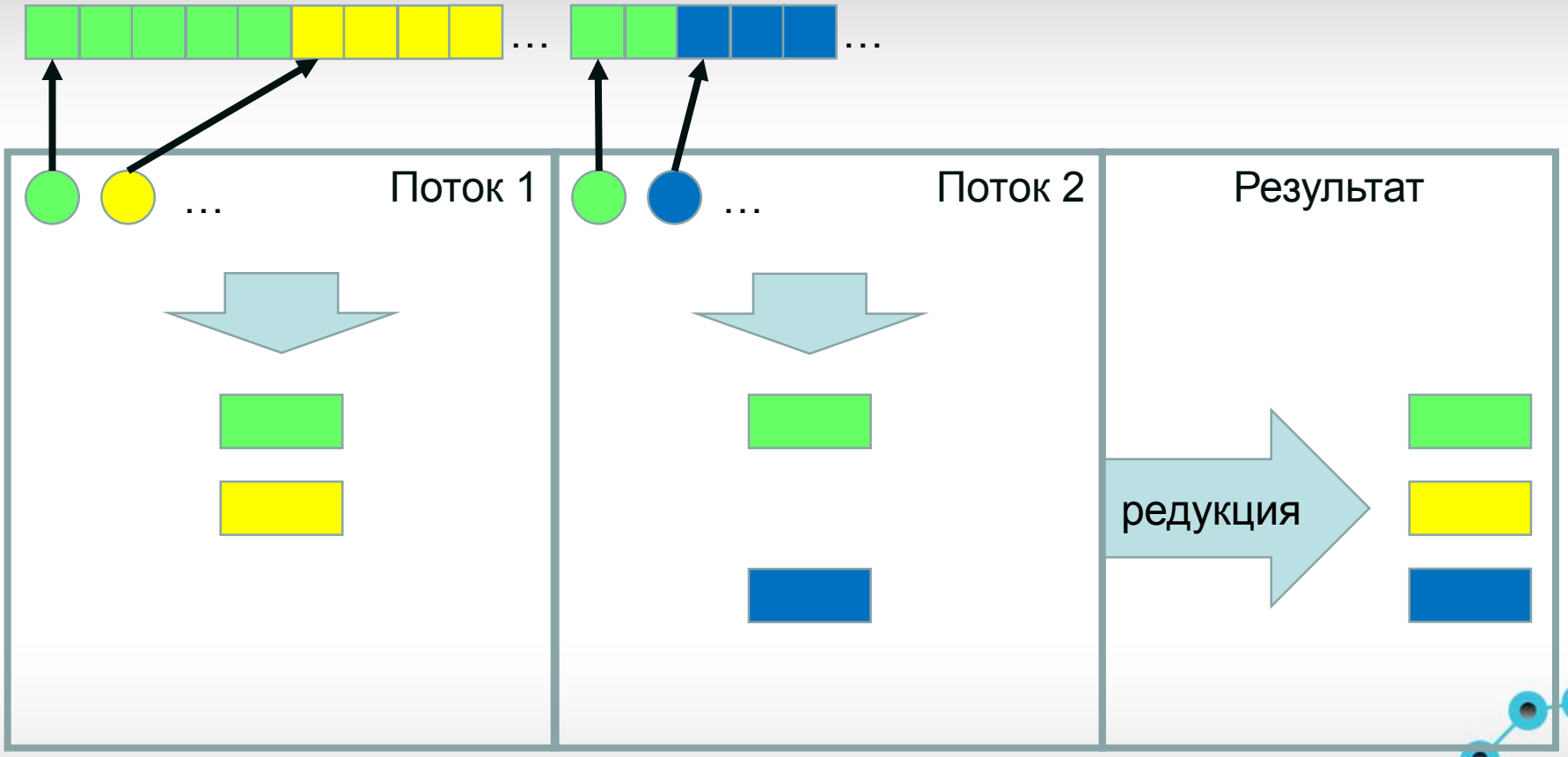
Массив рёбер: алгоритм

Множество рёбер графа хранится **одним массивом**.

На каждой итерации алгоритма Борувки:

a. каждый поток для **каждой** компоненты находит инцидентное ей ребро минимального веса
b. для каждой компоненты определяется ребро минимального веса по всем потокам
   редукция T массивов длины |V|
c. объединение компонент по найденным рёбрам
d. Pointer Jumping

Массив рёбер: представление в памяти

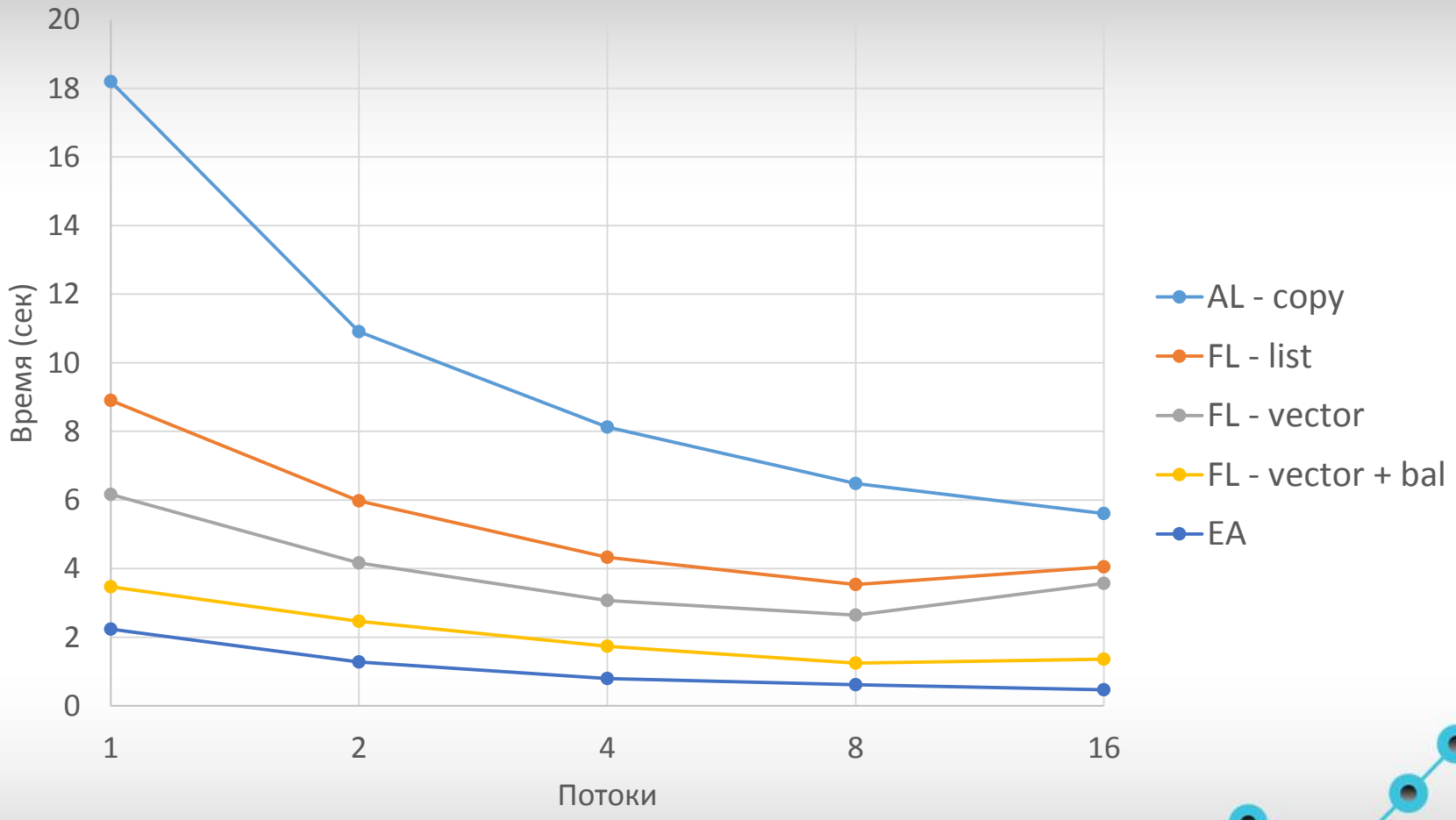


Массив рёбер: проблемы

a. каждый поток для каждой компоненты находит инцидентное ей ребро минимального веса
b. для каждой компоненты определяется ребро минимального веса по всем потокам
   редукция T массивов длины |V|
c. объединение компонент по найденным рёбрам
d. Pointer Jumping

Шаг (b) сложно масштабируем.

Массив рёбер: результаты



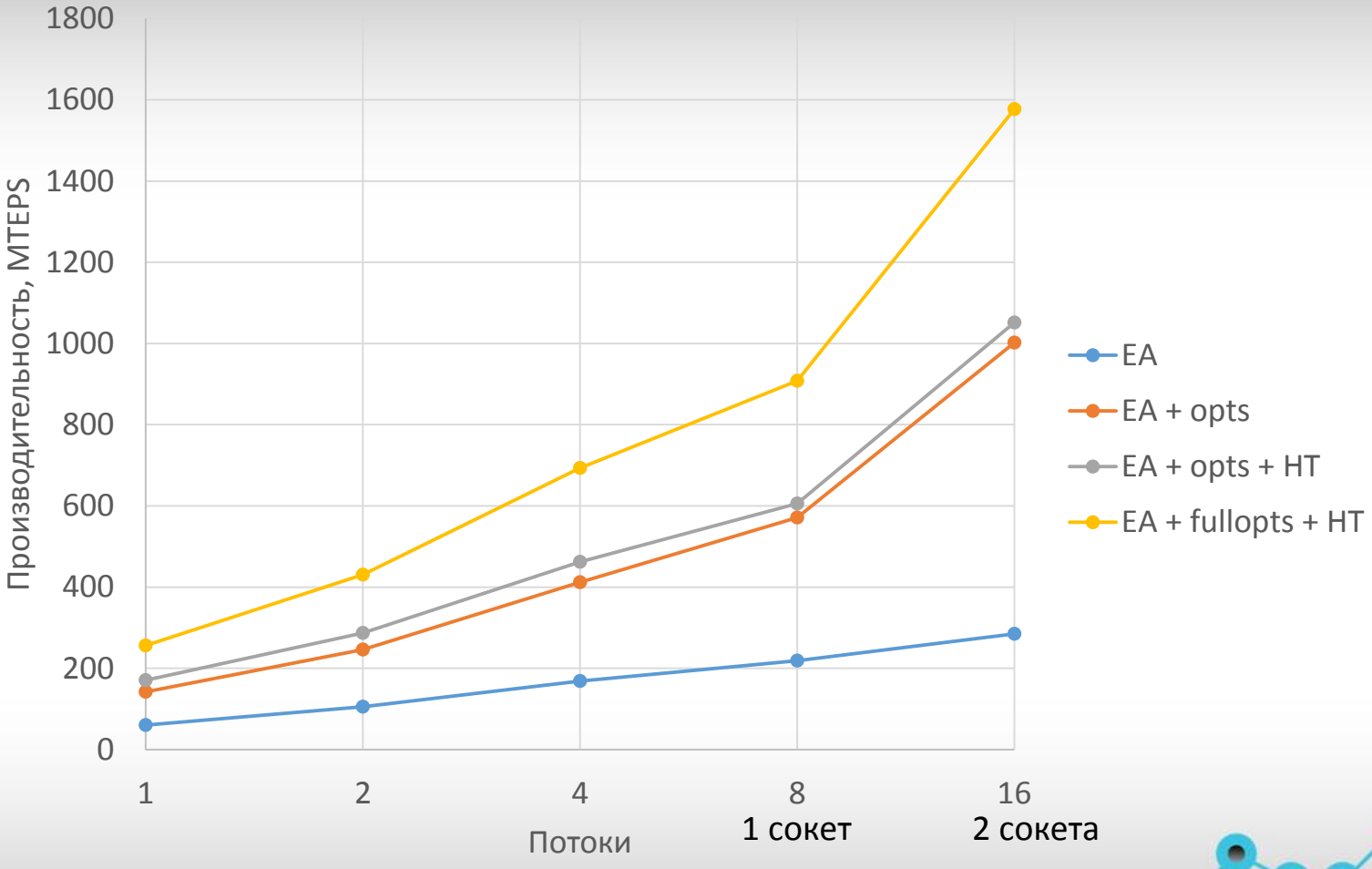
Массив рёбер: оптимизация редукции

- Отсутствие редукции на первой итерации (~20%)
  - Поскольку каждая компонента представлена ровно одной вершиной, то редукция на первой итерации избыточна
- Перенумерация компонент по ходу исполнения алгоритма (~20%)
  - Активные компоненты графа перенумеровываются так, чтобы их номера были последовательны
- Иерархическая редукция, NUMA (~10%)
  - Выделение памяти на тех сокетах, где она чаще используется
  - Редукция происходит в два этапа: внутри сокета и между сокетами.

Массив рёбер: оптимизации

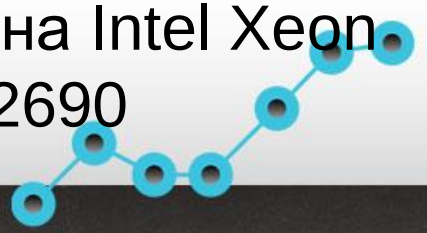
- Переупорядочивание вершин, балансировка вершин с большой степенью (~11%)
  - Вершины небольшой степени переупорядочиваются с помощью алгоритма Reverse Cuthill-McKee
  - Вершины высокой степени равномерно распределяются между потоками
- Отдельный цикл пропуска петель (~10%)
- Программная предвыборка (~30%)
- Hyper Threading (5% на 16 ядрах)
- Linux HugePages (5% на Intel Xeon Phi)

Массив рёбер: результаты



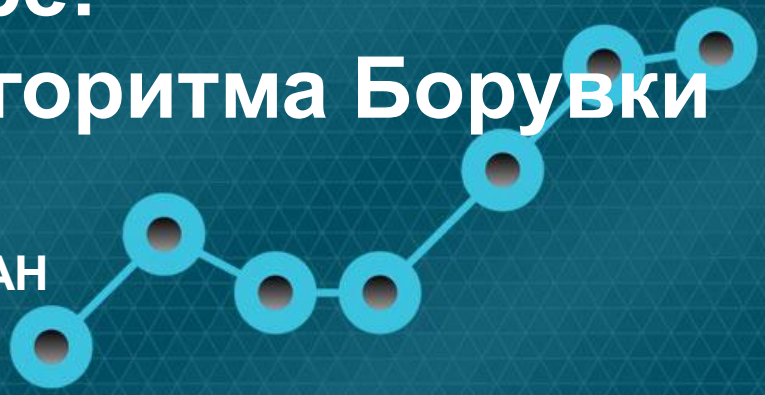
Результаты

- Были исследованы и реализованы 3 различных подхода к хранению графа в алгоритме Борувки.
- Общие рекомендации:
  - Чем меньше перестраивается граф во время исполнения, тем лучше производительность.
  - Программная предвыборка.
  - Упрощение глобальных (т.е. обходящих все вершины) циклов – их логика должна быть максимально простой.
- Первое место на больших графах на конкурсе GraphHPC (5 марта 2015, МГУ).
- Производительность текущей реализации на Intel Xeon Phi на уровне одного сокета Intel Xeon E5-2690



MST на мультипроцессоре: сравнение вариантов алгоритма Борувки

Зайцев Вадим, Новосибирский ГУ
Калгин Константин, к.ф.-м.н., ИВМиМГ СО РАН



Литература

- Frank Dehne, Silvia Gotz. *Practical parallel algorithms for minimum spanning trees*. http://people.scs.carleton.ca/~dehne/publications/2-47.pdf
- David A. Bader, Guojing Con. *Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs*. http://www.cc.gatech.edu/~bader/papers/MST-JPDC.pdf
- Sun Chung, Anne Condon. *Parallel implementation of Boruvka Minimum Spanning Tree Algorithm*. http://minds.wisconsin.edu/bitstream/handle/1793/60020/TR1297.pdf
- Guy E. Blelloch, Bruce M. Maggs. *Parallel Algorithms*. http://homes.cs.washington.edu/~arvind/cs424/readings/BlellochM96b.pdf