Система параллельной обработки графов для задач науки о сетях

ЧЕРНОСКУТОВ МИХАИЛ

ИММ УРО РАН, УРФУ

E-MAIL: MACH@IMM.URAN.RU

Содержание

Подходы к обработке графов
- Vertex-centric
- Domain-specific languages
- Gather-Apply-Scatter

Network Science и специфика данной области науки

Предлагаемый подход к построению системы (параллельной) обработки графов
- Эксперименты с SMP-системой
- Эксперименты с MPP-системой

Достоинства, недостатки предлагаемого подхода, планы на будущее

Современные системы обработки графов

Parallel Boost Graph Library, Pregel, CuSha, GraphCT, NetworkX, PowerGraph, graph-tool, GraphBLAS, KDT, igraph, STINGER, Ligra, Gunrock, HelP, GPS, Galois, Green-Marl, Gephi, Medusa, MapGraph, NetworKit, SNAP, GraphLab, Giraph, JUNG, Pajek, GraphPad, PEGASUS, GraphX, GraphChi, Totem, Vertexapi2

Классификация современных систем обработки графов

По поддержке параллелизма

- Без поддержки параллелизма (Gephi, NetworkX, …)
- SMP-параллелизм (GraphCT, GraphChi, …)
- MPP-параллелизм (Pregel, Parallel BGL, …)
- Использование ускорителей вычислений (Gunrock, MapGraph, …)

По модели обработки

- Vertex-centric
- Domain-specific languages
- Использование примитивов

Модель Vertex-centric

Каждая вершина содержит некоторые данные (о себе, о входящих и исходящих ребрах и другие необходимые данные)

Каждая вершина выполняет

- Прием и передачу сообщений по входящим и исходящим ребрам
- Некоторые локальные вычисления над данными, к которым у нее имеется доступ

Достоинство

- Позволяет естественным образом распараллеливать вычисления

Недостаток

- Сложности в реализации ряда алгоритмов (например, операции с матрицей смежности, инцидентности)

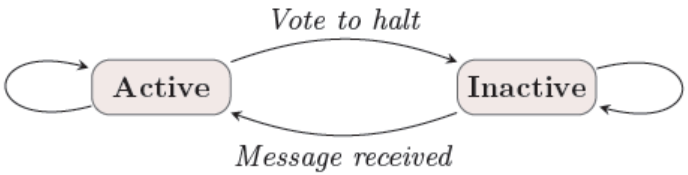
Модель Vertex-centric

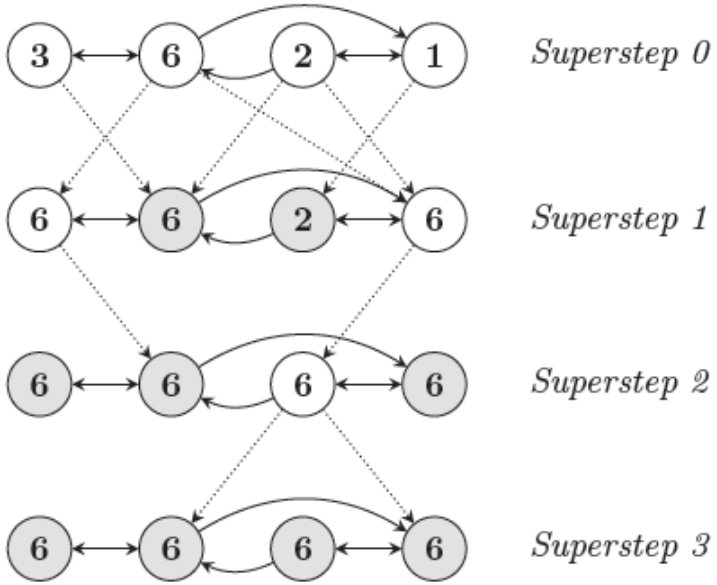
Первая реализация

- Pregel (Google, 2010 г.)

Другие реализации

- HelP, GPS и др.





Domain-Specific Language

Domain-Specific Language (DSL) – язык программирования, предназначенный для применения в определенной предметной области и содержащий специфические для нее выражения и конструкции

Достоинства

- Повышается продуктивность разработки
- Кроссплатформенность

Недостаток

- Сложности с интеграцией кода в уже имеющиеся проекты

Domain-Specific Language

Green-Marl – DSL для обработки графов на системах с общей памятью

- Содержит встроенный компилятор на C/C++

Другие реализации

- GraphChi, GraphLab

```
1  Procedure Compute_BC(
2   G: Graph, BC: Node_Prop<Float>(G)) {
3    G.BC = 0;        // initialize BC
4    Foreach(s: G.Nodes) {
5     // define temporary properties
6      Node_Prop<Float>(G) Sigma;
7      Node_Prop<Float>(G) Delta;
8      s.Sigma = 1; // Initialize Sigma for root
9     // Traverse graph in BFS-order from s
10     InBFS(v: G.Nodes From s)(v!=s) {
11       // sum over BFS-parents
12       v.Sigma = Sum(w: v.UpNbrs) {w.Sigma};
13     }
14    // Traverse graph in reverse BFS-order
15     InRBFS(v!=s) {
16       // sum over BFS-children
17       v.Delta = Sum (w:v.DownNbrs) {
18          v.Sigma / w.Sigma * (1+ w.Delta)
19       };
20       v.BC += v.Delta @s; //accumulate BC
21 } } }
```

Примитивы обработки

Идея заключается в выделении обобщенных операций для широкого класса алгоритмов на графах и их параллельная реализация в виде отдельных функций

Алгоритмы на графах реализуются в виде различных комбинаций примитивов

Достоинства

- Упрощение разработки и отладки
- Совместное использование с любым другим кодом

Недостаток

- Пока не найден полный набор примитивов, с помощью которых можно реализовать любой алгоритм на графах

Примитивы обработки

GraphBLAS

- Попытка описания алгоритмов на графах в термина операций с линейной алгеброй
- Разработка ведется с 2008 г. отдельными научными группами и крупными компаниями (IBM, Intel)

```
#include <stdio.h>
#include "gpi.h"

GPI_int32 BFS(GPI_Matrix *A, GPI_index s, GPI_Function1* visit)
/*
 * Given an adjacency matrix A and a source node s, performs a BFS traversal
 *   and calls "visit" on the vertices visited (excluding source).
 */
{
  GPI_uint32 n; GPI_Matrix_nrows(&n, A);
  GPI_Vector q, p, r, v;

  GPI_Vector_new(&q, GPI_int32, n);
  GPI_replicate(&q, GPI_zero);
  GPI_Vector_setElement(&q,s,GPI_one);
  GPI_Vector_new(&p, GPI_int32, n);
  GPI_Vector_copy(&p,&q);
  GPI_Matrix B; GPI_transpose(&B,A);
  GPI_Vector_new(&r, GPI_int32, n);
  GPI_Vector_new(&v,GPI_int32,n);
  GPI_indices(&v);

  /*
   * BFS traversal and visits the vertices.
   */
  bool done = false;
  do {
    GPI_replicate(&r, GPI_zero);
    GPI_mxv(&r,&B,&q,GPI_LOR,GPI_LAND);
    GPI_filter(&q,GPI_zero,&p,&r);
    GPI_zip(&p,GPI_LOR,&p,&q);
    GPI_map(&r,visit,&v,&q);
    GPI_Scalar sum;
    GPI_reduce(&sum,GPI_LOR,GPI_zero,&q);
    done = (sum == GPI_zero);
  } while (!done);

  GPI_Vector_delete(&q); GPI_Vector_delete(&p);
  GPI_Vector_delete(&r); GPI_Vector_delete(&v);

  return 0;
}
```

Возможности современных систем обработки графов

По размеру обрабатываемых графов (данные из открытых источников)

- Pregel – 127 млрд. вершин
- Parallel BGL – 17 млрд. вершин
- KDT – 17 млрд. вершин
- NetworkX – 100 млн. вершин
- igraph – неск. млн. вершин

По количеству реализованных алгоритмов

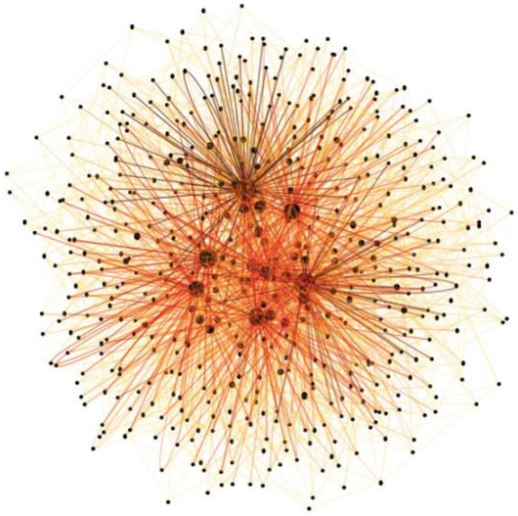
- Наибольшее количество реализованных алгоритмов – в системах, ориентированных на последовательную обработку (в т.ч. с использованием скриптовых языков)
- В системах, ориентированных на массивно-параллельную обработку, алгоритмов реализовано гораздо меньше

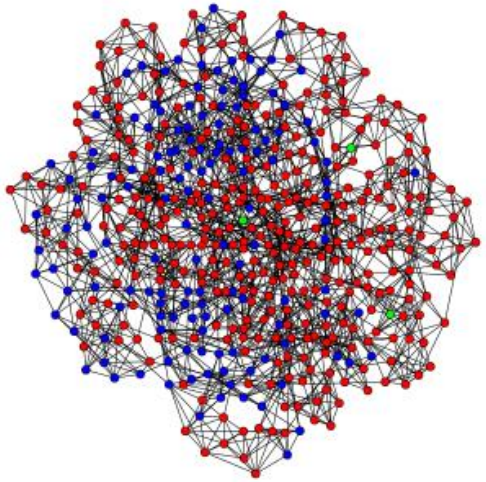
Network Science

Область науки, изучающая комплексные сети

Комплексные сети (Complex Networks) – сети, обладающие нетривиальными топологическими свойствами (связи между вершинами не случайны и часто могут подчиняться какому-либо закону)

- Scale-free и Small-world





Network Science

Примеры задач из Network Science

- Анализ структуры протеинов
- Предсказание новых связей в социальных сетях
- Моделирование активности мозга
- Исследование городского траффика

Примеры алгоритмов, используемых в Network Science

- Поиск сообществ
- Поиск одинаковых базовых элементов
- Распространение сигналов по сети
- Исследование многоуровневых сетей / мультиграфов / гиперграфов
- Анализ устойчивости сетей и др.

Использование имеющихся систем обработки графов при решении задач Network Science весьма ограничено

Проблема выбора структуры данных для работы с графом

Пусть необходимо разработать и реализовать новый алгоритм с помощью какой-либо из имеющихся систем обработки графов

Возможные проблемы

- Затрудненная навигация по графу
- Затрудненная модификация топологии графа
- Нелинейный рост объема памяти, требующейся для хранения графа
- Невозможность использования определенных типов графов (например, с параллельными ребрами)

Проблема выбора структуры данных для работы с графом

Пример – поиск максимального потока в сети

- Затруднен быстрый поиск ребер в структуре данных CSR
- Матрица смежности требует большого объема доступной памяти

FORD_FULKERSON(G, s, t)

1 **for** (для) каждого ребра $(u, v) \in E[G]$
2 **do** $f[u, v] \leftarrow 0$
3 $f[v, u] \leftarrow 0$
4 **while** существует путь p из s в t в остаточной сети G_f
5 **do** $c_f(p) \leftarrow \min \{c_f(u, v) : (u, v)$ принадлежит $p\}$
6 **for** (для) каждого ребра (u, v) in p
7 **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
8 $f[v, u] \leftarrow -f[u, v]$

Система параллельной обработки графов

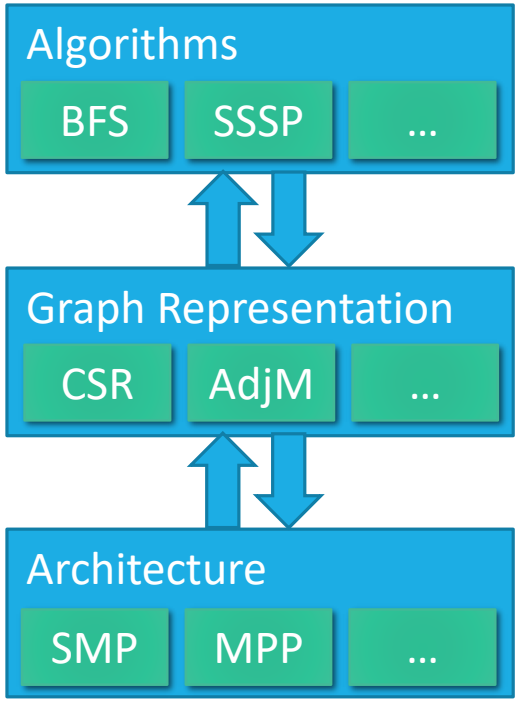
Мотивация создания новой системы

- Упрощение разработки алгоритмов
  - Переход от операций с массивами, матрицами, переменными к операциям со структурой данных графа
  - Использование в коде конструкций, напоминающих псевдокод алгоритма
- Независимость от структуры данных
  - Возможность выбрать для каждого алгоритма наиболее удобное представление графа в памяти
  - Возможность разрабатывать новые, более оптимальные, представления графа в памяти вычислительной системы
- Независимость от используемой архитектуры
  - Возможность использования как SMP-, так и MPP-систем

Система параллельной обработки графов

Предлагаемая архитектура системы

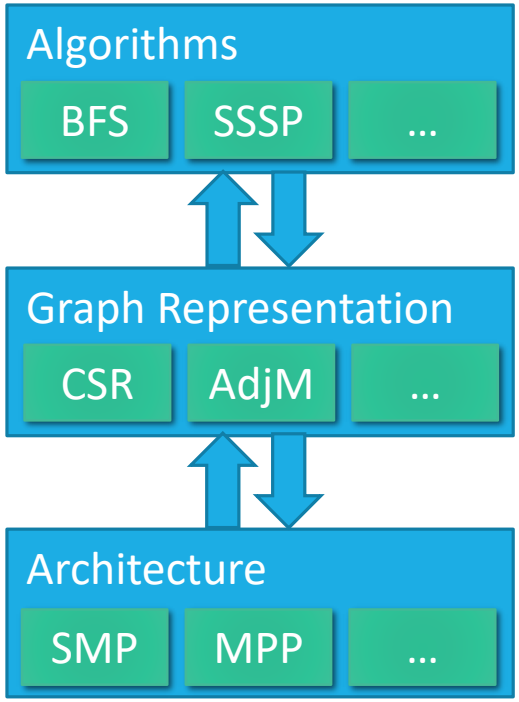
- Абстрактный класс, содержит абстрактные методы для работы с графом
- Каждый класс-потомок реализует абстрактные методы, позволяя реализовать требуемую структуру данных
- Алгоритмы реализуются за счет операций с объектом, представляющим требуемую для алгоритма структуру данных
- Для разработки новой структуры данных необходимо создать новый класс-потомок, содержащий новый способ реализации базовых методов для работы с графом



Система параллельной обработки графов

Примеры базовых функций для работы с графом

- `is_directed`
- `is_weighted`
- `get_number_of_nodes`
- `get_number_of_edges`
- `get_ingoing_neighbors`
- `get_outgoing_neighbors`
- `set_edge_weight`
- `add_node` / `delete_node`
- `add_edge` / `delete_edge`
- И другие…



Система параллельной обработки графов

SMP-параллелизм

- Непосредственно с помощью стандарта OpenMP

MPP-параллелизм

- Посредством MPI с помощью встроенных в систему функций

- Используются специальные объекты-накопители данных
- Перед работой со свойствами вершин (номер уровня, значение метрики центральности, величина потока через вершину и т.д.) необходимо “накопить” на вычислительном узле данные о вершинах-соседях, которые могут находиться на других вычислительных узлах
- После работы с “накопленными” данными необходима синхронизация

- При работе на SMP-системе этапы синхронизации и накопления данных на узле могут быть пропущены за ненадобностью

Система параллельной обработки графов

```
vector<node_id> neighb; int some_var;
      // Объявление объекта-накопителя данных
node_prop some_prop;
      // Сбор заявок
for(node_id i=0; i<num_of_nodes; i++)
      some_prop.add(graph.get_outgoing_neighbors(i));
synch_prop_get_data(some_prop);
      // Работа с данными
for(node_id i=0; i< num_of_nodes; i++)
{
      neighb = graph.get_outgoing_neighbors(i);
      for(node_id j=0; j<neighb.size(); j++)
            some_prop.set(neighb[j], some_var);
}
synch_prop_set_data(some_prop);
```

Система параллельной обработки графов

Место предлагаемого подхода в имеющемся наборе ПО для обработки больших графов

Степень параллелизма

Сложность разработки	**SMP**	**MPP (до 50 узлов)**	**MPP (более 50 узлов)**
Просто	NetworkX, igraph	ND	ND
Умеренно	ND		GraphBLAS
Сложно	Самостоятельная разработка на C/C++	ND	Pregel, Parallel BGL

Пример (поиск максимального потока)

Две разных структуры данных

- Модифицированная структура данных CSR для работы с параллельными ребрами
  - Каждому ребру присваивается тег, позволяющих отличать друг от друга параллельные ребра
- Набор матриц смежности для хранения значений потока на ребрах
  - Матрица для “прямого” и “обратного” потока по ребру в исходном графе

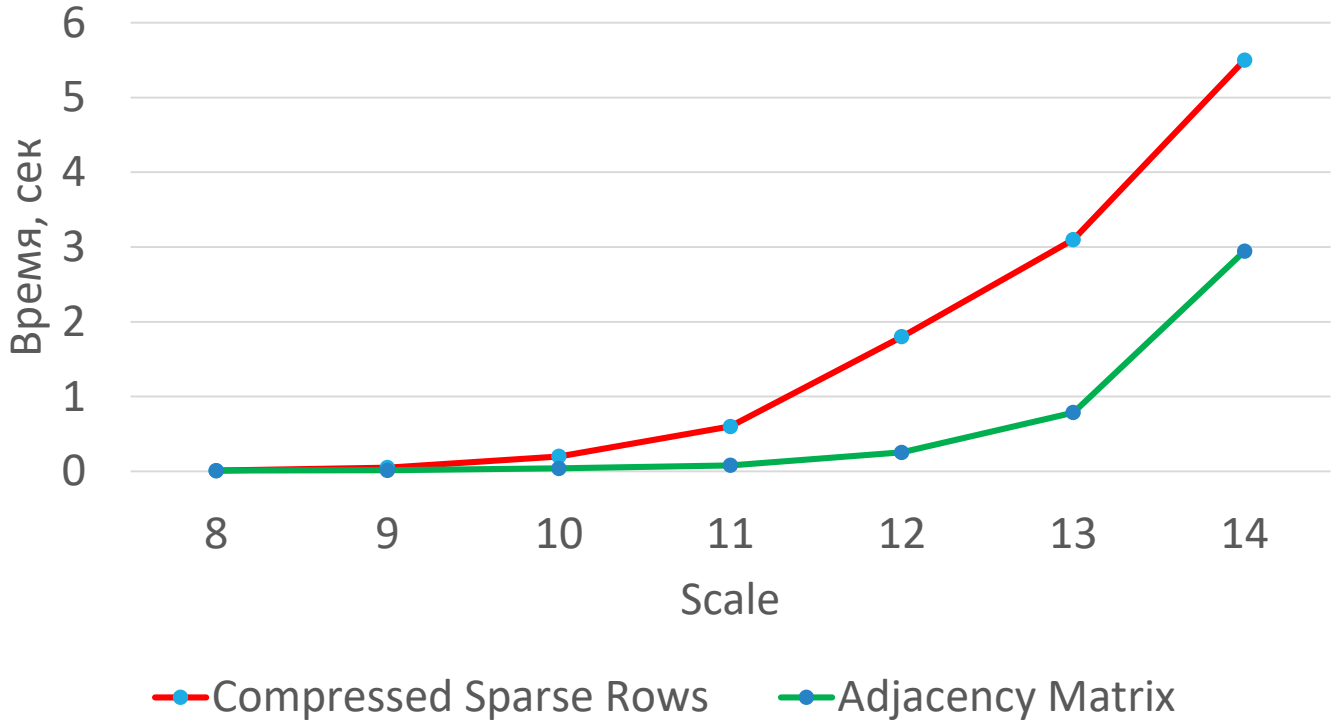
Эксперименты проводились с использованием SMP-системы

Аппаратное обеспечение

- 2 x Intel Xeon X5675
- 46 ГБ ОЗУ

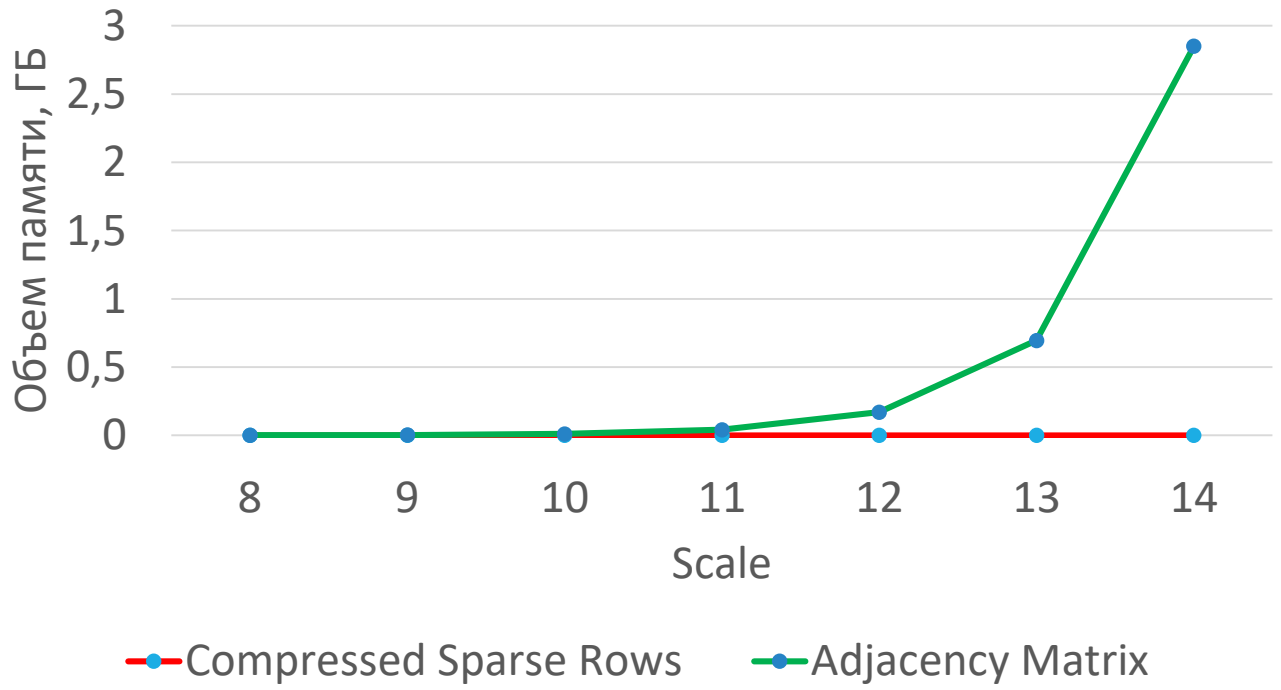
Пример (поиск максимального потока)

Скорость обработки графа



Пример (поиск максимального потока)

Количество потребляемой памяти



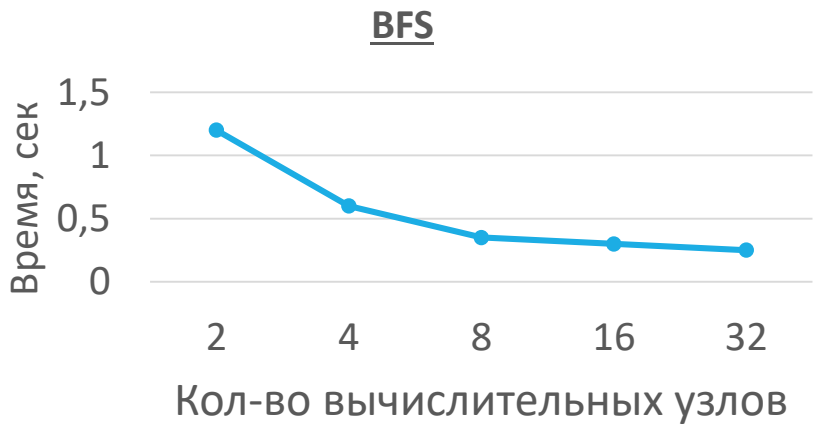
Пример (поиск в ширину, поиск кратчайших путей)

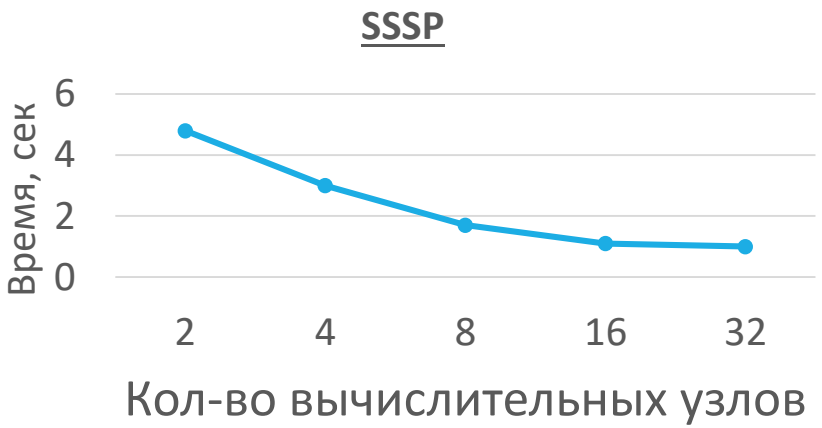
Эксперименты проводились с использованием MPP-системы

Эксперименты проводились с графом, имеющим 2^{20} вершин

Аппаратное обеспечение

- 2 x Intel Xeon E5450
- 16 ГБ ОЗУ
- 20 Гбит IB DDR





Примеры прикладных задач (анализ ж/д расписаний)

Задача – построение графа рейсов ж/д транспорта

Подзадачи

- Анализ графа ж/д сети в рамках страны/региона
- Поиск путей в графе ж/д сети
- Построение графа связей между отдельными путями в графе ж/д сети
  - Получившийся граф имеет параллельные ребра
- Перестроение графа связей в случае изменения расписания движения ж/д транспорта (отмена, задержка и т.д.)

Примеры прикладных задач (снятие лексической многозначности)

Задача – кластеризация графа синонимов с целью выделения групп близких по значению слов (синсетов)

Подзадачи

- Выделение “эго-сетей” различного порядка
- Анализ выделенных “эго-сетей” различными алгоритмами (коэффициент кластеризации, перебор ребер, поиск кратчайших путей, поиск клик)
- Сравнение “эго-сетей” между собой на основании выполненного анализа

Достоинства и недостатки предлагаемого подхода

Достоинства

- Возможность разработки и сравнительного исследования новых структур данных для хранения графов
- Относительная простота разработки алгоритмов (не нужно напрямую работать с функциями MPI и управлять перемещением данных)
- Возможность запускать один и тот же код для SMP- и MPP-систем

Недостатки

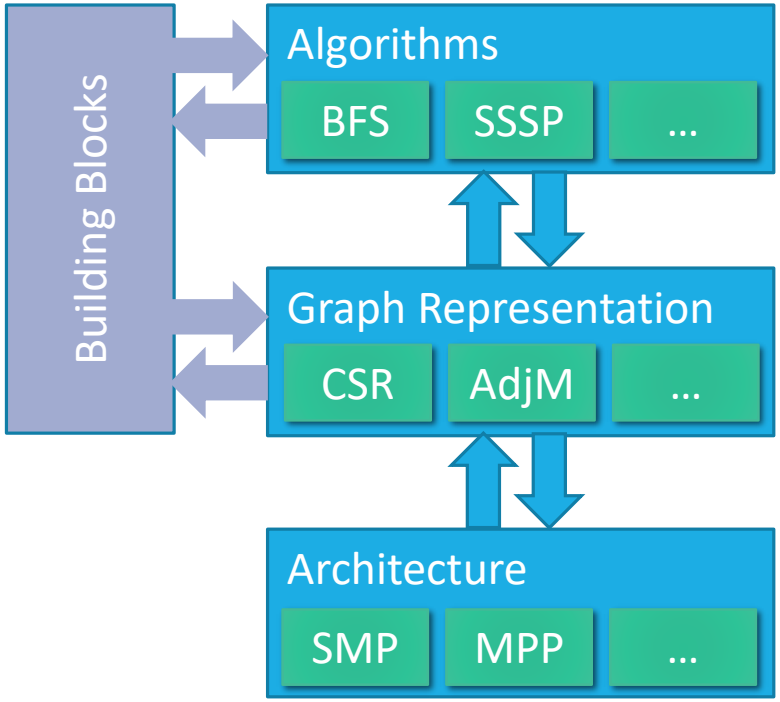
- Относительно низкая производительность при работе с MPP-системами
- При наличии ветвления в коде не поддерживается работа с объектами-накопителями

Планы на будущее

Поиск новых задач

Повышение масштабируемости на SMP- и MPP-системах

Разработка набора примитивов для обработки графов



Вопросы?