Библиотека Galois

- ***Galois*** — это библиотека C++, предназначенная для упрощения параллельного программирования,
  особенно для приложений с нерегулярным параллелизмом (например, нерегулярным объемом работы в параллельных разделах, нерегулярным доступом к памяти и шаблонами ветвления).

- Библиотека реализует модель неявно параллельного программирования, в которой программист заменяет конструкции последовательного цикла (например, for и while) и последовательные структуры данных в своих алгоритмах конструкциями параллельного цикла и параллельными структурами данных, предоставленными Galois.

- ***Galois*** спроектирован таким образом, чтобы программисту не приходилось иметь дело с низкоуровневыми конструкциями параллельного программирования, такими как: потоки, блокировки, условные переменные и т.д.

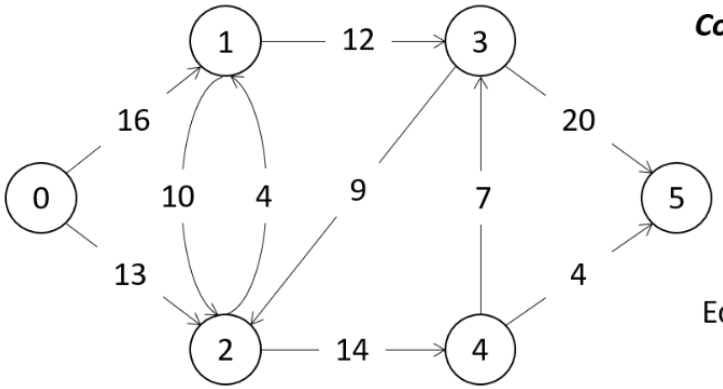
ПРЕДСТАВЛЕНИЕ ГРАФОВ В БИБЛИОТЕКЕ

Графы

- galois::graphs::MorphGraph: позволяет вставлять и удалять узлы и ребра.

- galois::graphs::LC_Morph_Graph можно использовать, если (1) ни один узел не будет удален и (2) максимальная степень узла известна при его создании. Внутри он реализован в формате сжатых разреженных строк. Неориентированные ребра представляются как два направленных ребра как два направленных.

Графы. LC_Morph_Graph



Compressed sparse row

Node data	n0	n1	n2	n3	n4	n5					
Edge index	0	2	4	6	8	10	10				
Edge destination	1	2	2	3	1	4	2	5	3	5	Ø
Edge data	16	13	10	12	4	14	9	20	7	4	

Graph in CSR Format

Графы. Программный интерфейс

- Ниже приведен пример определения LC_CSR_Graph с целым числом в качестве типа данных узла и типа данных ребра:

```
using Graph = galois::graphs::LC_CSR_Graph<int, int>;
```

- Читаем граф из файла (в двоичном формате gr):

```
galois::graphs::readGraph(g, argv[1]); // argv[1] is the file name for graph
```

- Чтобы прочитать/записать данные узла

```
galois::graphs::LC_CSR_Graph::getData
```

- Чтобы прочитать/записать данные о ребре

```
galois::graphs::LC_CSR_Graph::getEdgeData
```

- Чтобы прочитать/записать данные о ребре

```
galois::graphs::LC_CSR_Graph::getEdgeDst
```

СОДЕРЖАНИЕ БИБЛИОТЕКИ

Содержание библиотеки

- Параллельный цикл ***for_each***, который обрабатывает зависимости между итерациями, а также динамическое создание работ, и цикл ***do_all*** для простого параллелизма.
- Библиотека параллельных графов, предназначенная для алгоритмов анализа графов.
- Масштабируемые параллельные контейнеры, такие как ***bag, vector, list*** и т. д.

ПАРАЛЛЕЛЬНЫЕ ЦИКЛЫ

Параллельные циклы. do_all

- galois::do_all равномерно распределяет рабочие элементы между доступными потоками. Он часто используется в таких случаях, как алгоритмы на основе топологии, которые перебирают узлы графа. Его также часто используют в случаях, когда необходимо выполнить группу независимых рабочих элементов.

- Обход графа. Последовательная версия.

```
Graph g;
 galois::graphs::readGraph(g, argv[1]); // argv[1] is the file name for graph

 for (auto n : g) {
   auto& sum = g.getData(n);
   sum       = 0;
   for (auto e : g.edges(n)) {
     sum += g.getEdgeData(e);
   }
 }
```

- Обход графа. Параллельная версия.

```
galois::do_all(
     galois::iterate(g.begin(), g.end()), // range
     [&](GNode n) {                       // operator
       auto& sum = g.getData(n);
       sum       = 0;
       for (auto e : g.edges(n)) {
         sum += g.getEdgeData(e);
       }
     },
     galois::loopname("sum_in_do_all_with_lambda") // options
 );
```

Параллельные циклы. for_each

- galois::for_each можно использовать для параллельных итераций, которые могут генерировать новые задачи и могут иметь конфликты между итерациями.

- Алгоритм push-стиля: каждый узел добавляет вес каждого своего ребра к соответствующему соседу.

```
initialize(g);
galois::for_each(
galois::iterate(g.begin(), g.end()), // range
[&](GNode n, auto&) { // operator
for (auto e : g.edges(n)) { // cautious point
auto dst = g.getEdgeDst(e);
g.getData(dst) += g.getEdgeData(e);
}
},
galois::loopname("sum_in_for_each_with_push_operator") // options
);
```

Параллельные циклы. for_each

- Дан граф G с ребрами с весом и без циклов с отрицательным весом и исходный узел s; Первоначально расстояние s от самого себя равно 0, а все остальные узлы находятся на расстоянии бесконечности от s.

```
auto SSSP = [&](GNode active_node, auto& ctx) {
                    // Get the value on the node
                    auto srcData = graph.getData(active_node);
                    // loop over neighbors to compute new value
                    for (auto ii : graph.edges(active_node)) { // cautious point
                                        auto dst = graph.getEdgeDst(ii);
                                        auto weight = graph.getEdgeData(ii);
                                        auto& dstData = graph.getData(dst);
                                        if (dstData > weight + srcData) {
                                                            dstData = weight + srcData;
                                                            ctx.push(dst); // add new work items
                                        }
                    }
};

 // clear source
  graph.getData(*graph.begin()) = 0;
      galois::for_each(
       galois::iterate(
          {*graph.begin()}), // initial range using initializer list
       SSSP                  // operator
       ,
       galois::wl<PSchunk>() // options. PSchunk expands to
                        // galois::worklists::PerSocketChunkLIFO<16>,
                        // where 16 is chunk size
       ,
       galois::loopname("sssp_dchunk16"));
  }
```

ВЫВОД ПРОГРАММЫ

Вывод программы

- По завершении приложения Galois будут выводить статистику в формате csv, аналогично следующему:

```
STAT_TYPE, REGION, CATEGORY, TOTAL_TYPE, TOTAL
STAT, for_each_1, Iterations, TSUM, 9028387
STAT, for_each_1, Time, TMAX, 1663
STAT, for_each_1, Commits, TSUM, 9000000
STAT, for_each_1, Pushes, TSUM, 0
STAT, for_each_1, Conflicts, TSUM, 28387
```

- REGION сообщает вам, к какому параллельному циклу относится статистика. Например, "for_each_1" относится к galois::for_each, у которого есть опция galois::loopname("for_each_1").
- CATEGORY указывает, что сообщается для параллельного региона. Для циклов galois::for_each выводятся следующие пять статистических данных:
  - Итерации: количество выполненных итераций.
  - Время: время выполнения в миллисекундах.
  - Коммиты: количество совершенных итераций.
  - Pushs: количество сгенерированных итераций.
  - Конфликты: количество итераций, прерванных из-за конфликтов.
- TOTAL_TYPE расскажет вам, как формируется статистика
- TSUM означает, что значение представляет собой сумму вкладов всех потоков;
- TMAX означает, что это максимум среди всех потоков для этой статистики

СПАСИБО ЗА ВНИМАНИЕ!