User guide по библиотеке Galois

Galois — это библиотека C++, разработанная для упрощения параллельного программирования, особенно для приложений с неравномерным параллелизмом. Galois разработан таким образом, что программисту не приходится иметь дело с низкоуровневыми конструкциями параллельного программирования, такими как потоки, блокировки, барьеры, переменные условий и т.д.

Основные возможности и особенности Galois:

1. Galois предоставляет средства для реализации параллельных циклов, которые автоматически управляют распределением задач между потоками:
    - параллельный цикл for_each, который обрабатывает зависимости между итерациями, а также динамическим созданием работ
    - параллельный цикл do_all для простого параллелизма.

   Оба цикла обеспечивают балансировку нагрузки и отличную масштабируемость на мультисокетных системах
2. Galois предоставляет различные структуры данных для работы с графами, включая структуры, оптимизированные для параллельной работы. Galois активно используется в задачах графовой аналитики.
3. Galois может автоматически выявлять и управлять зависимостями данных, минимизируя необходимость ручной синхронизации и блокировки, которые часто приводят к ошибкам в многопоточных приложениях.

Сборка библиотеки:

Galois собирается, запускается и тестируется на GNU/Linux.

Библиотека зависит от следующего программного обеспечения:

- Современный компилятор C++, совместимый со стандартом C++-17 (gcc >= 7, Intel >= 19.0.1, clang >= 7.0)
- CMake (>= 3.13)

- Дополнительные библиотеки: могут включать Boost (>= 1.58.0) или LLVM (>= 7.0 с поддержкой RTTI), в зависимости от конфигурации и используемых функций Galois

Шаги сборки:

1. git clone https://github.com/IntelligentSoftwareSystems/Galois
2. SRC_DIR=`pwd` # or top-level Galois source dir
3. BUILD_DIR=<path-to-your-build-dir>
4. mkdir -p $BUILD_DIR
5. cmake -S $SRC_DIR -B $BUILD_DIR -DCMAKE_BUILD_TYPE=Release
6. make-jN # N —число параллельных потоков компиляции
7. make install
8. make test

Запуск тестов может быть полезен для проверки корректности сборки:

```
Start   1: unit-getenv
Test   #1: unit-getenv ..........................................   Passed    0.00 sec
Start   2: unit-logging
Test   #2: unit-logging .........................................   Passed    0.00 sec
Start   3: unit-acquire
Test   #3: unit-acquire .........................................   Passed    0.00 sec
Start   4: unit-bandwidth
Test   #4: unit-bandwidth .......................................   Passed    0.08 sec
Start   5: unit-barriers
Test   #5: unit-barriers ........................................   Passed    0.02 sec
Start   6: unit-empty-member-lcgraph
Test   #6: unit-empty-member-lcgraph ............................   Passed    0.00 sec
Start   7: unit-flatmap
Test   #7: unit-flatmap .........................................   Passed    0.01 sec
Start   8: unit-floatingPointErrors
Test   #8: unit-floatingPointErrors .............................   Passed    0.00 sec
Start   9: unit-foreach
Test   #9: unit-foreach .........................................   Passed    0.02 sec
Start  10: unit-forward-declare-graph
Test  #10: unit-forward-declare-graph ...........................   Passed    0.01 sec
```

После успешной сборки и установки можно использовать библиотеку в своих проектах.

Работа с графами:

В библиотеке графы хранятся в двоичном формате, называемом galois graph file (расширение файла .gr). Другие форматы, такие как edge-list или Matrix-

Market, могут быть преобразованы в формат .gr с помощью инструмента graph-convert. Можно создать graph-convert следующим образом:

```
cd $BUILD_DIR
make graph-convert
./tools/graph-convert/graph-convert –help
```

Далее для демонстрации примеров работы с библиотекой запустим ее на двух задачах.

Алгоритм дельта-шага (Delta-Stepping) для нахождения кратчайших путей в графе

Идея: модифицируем алгоритм Дейкстры так, чтобы выделить группы вершин, для которых расстояние можно пересчитать одновременно.

Выберем порог $\Delta > 0$. Все ребра по весу можно разделить на «легкие» ($\{(v,u) \in E : w(v,u) < \Delta\}$) и «тяжелые» $\{(v,u) \in E : w(v,u) \leq \Delta\}$. Для хранения оценки расстояния используем массив карманов (buckets), в которых вершины сгруппированы по текущей оценке расстояния. В кармане $B[i]$ хранятся вершины v из очереди, для которых оценка расстояния $d[v] \in [\ i\Delta,(i+1)\Delta)$. Величина Δ называется шириной кармана.

Если вершины связаны «легким» ребром, то при пересчете расстояния они окажутся в одном кармане. Если вершины связаны «тяжелым» ребром, то они окажутся в разных карманах. Одновременно можно пересчитать расстояние по всем «легким» ребрам, исходящим из вершин одного кармана. В результате этой операции в текущий карман могут попасть новые вершины или вершины, которые уже были в нем, но с улучшенной оценкой расстояния.

1. *Предобработка*. Для всех $v \in V$: разделить список смежности вершины по весу ребер: $L[v]$ – все ребра с весом меньше Δ, $H[v]$ – все ребра с весом больше Δ.
2. *Инициализация*. Для всех $v \in V$: $d[v] = \infty$. $B[0] = \{s\}$. $i = 0$.
3. Пока все карманы *B* не пусты:
    1. Множество рассмотренных вершин $S = \emptyset$
    2. Для каждого кармана $B[i]$ выполнять, пока карман не пуст:
        1. Сформировать множество пар Req, состоящее из вершин, соединенных с вершинами из $B[i]$ легким ребром, и новой оценкой расстояния для них: $Req = \{(u,x)\colon v \in B[i], (v,u) \in L(v), x = d[v] + w(v,u)\}$.
        2. $S = S \cup B[i]$; $B[i] = \emptyset$
        3. *Релаксация*. Обработать Req: для каждого $(u,x) \in Req$ если $d[u] > x$, то $d[u] = x$, исключить u из кармана $B[d[u] / \Delta]$, вставить u в карман $B[x / \Delta]$.
    3. Сформировать множество пар вершин, соединенных с вершинами из *S* тяжелым ребром, и новой оценки расстояния для них:
    $$Req = \{(u,x)\colon v \in B[i], (v,u) \in H(v), x = d[v] + w(v,u)\}$$
    4. *Релаксация.* Обработать Req: для каждого $(u,x) \in Req$ если $d[u] > x$, то $d[u] = x$, исключить u из кармана $B[d[u]/\Delta]$, вставить u в карман $B[x/\Delta]$.

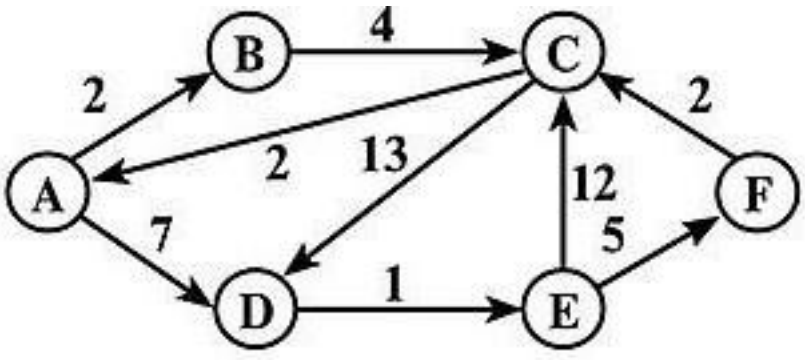
Код параллельной реализации с использованием Galois:

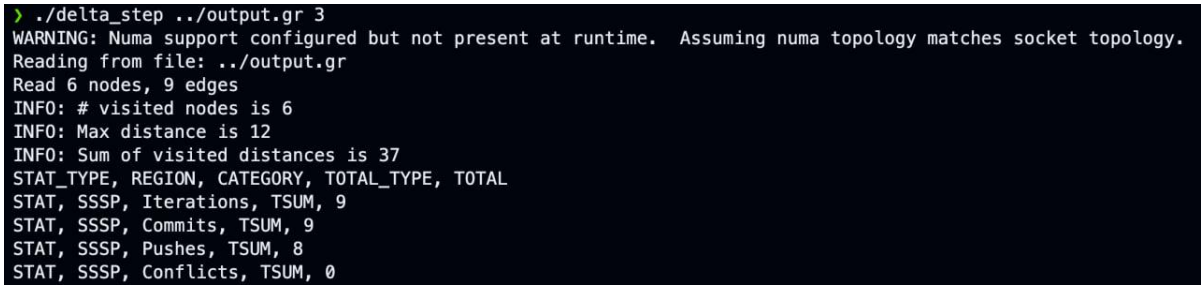
```
#include "galois/Galois.h"
#include "galois/AtomicHelpers.h"
#include "galois/Reduction.h"
#include "galois/Timer.h"
#include "galois/graphs/LCGraph.h"
#include "galois/graphs/FileGraph.h"
#include "Lonestar/BFS_SSSP.h"

#include <functional>
#include <iostream>
#include <string>
#include <limits>
#include <vector>
#include <queue>

constexpr unsigned int stepShift = 13;
constexpr unsigned int CHUNK_SIZE = 16;
constexpr static const ptrdiff_t EDGE_TILE_SIZE = 512;

using Graph = galois::graphs::LC_CSR_Graph<std::atomic<uint32_t>, uint32_t>::
   with_no_lockable<true>::type;
typedef Graph::GraphNode GNode;

using SSSP              = BFS_SSSP<Graph, uint32_t, true, EDGE_TILE_SIZE>;
```

```
using Dist               = SSSP::Dist;
using UpdateRequest      = SSSP::UpdateRequest;
using UpdateRequestIndexer = SSSP::UpdateRequestIndexer;
using ReqPushWrap        = SSSP::ReqPushWrap;
using OutEdgeRangeFn     = SSSP::OutEdgeRangeFn;

namespace gwl = galois::worklists;
using PSchunk = gwl::PerSocketChunkFIFO<CHUNK_SIZE>;
using OBIM    = gwl::OrderedByIntegerMetric<UpdateRequestIndexer, PSchunk>;
using OBIM_Barrier =
   gwl::OrderedByIntegerMetric<UpdateRequestIndexer,
                               PSchunk>::with_barrier<true>::type;

template <typename T, typename OBIMTy = OBIM, typename P, typename R>
void deltaStepAlgo(Graph& graph, GNode source, const P& pushWrap,
                   const R& edgeRange) {

  graph.getData(source) = 0;

  galois::InsertBag<T> initBag;
  pushWrap(initBag, source, 0, "parallel");

  galois::for_each(
      galois::iterate(initBag),
      [&](const T& item, auto& ctx) {
        constexpr galois::MethodFlag flag = galois::MethodFlag::UNPROTECTED;
        const auto& sdata                 = graph.getData(item.src, flag);

        if (sdata < item.dist) {
          return;
        }

        for (auto ii : edgeRange(item)) {

          GNode dst         = graph.getEdgeDst(ii);
          auto& ddist       = graph.getData(dst, flag);
          Dist ew           = graph.getEdgeData(ii, flag);
          const Dist newDist = sdata + ew;
          Dist oldDist      = galois::atomicMin<uint32_t>(ddist, newDist);
          if (newDist < oldDist) {
            pushWrap(ctx, dst, newDist);
          }
        }
      },
```

```
      galois::wl<OBIMTy>(UpdateRequestIndexer{stepShift}),
      galois::disable_conflict_detection(), galois::loopname("SSSP"));
}

int main(int argc, char** argv) {
  if (argc < 2) {
    std::cerr << "Usage: " << argv[0] << " <input graph file> [startNode]\n";
    return 1;
  }

  std::string inputFile = argv[1];
  unsigned int startNode = argc > 2 ? std::stoi(argv[2]) : 0;

  galois::SharedMemSys G;
  Graph graph;

  std::cout << "Reading from file: " << inputFile << "\n";
  galois::graphs::readGraph(graph, inputFile);
  std::cout << "Read " << graph.size() << " nodes, " << graph.sizeEdges() << " edges\n";

  if (startNode >= graph.size()) {
    std::cerr << "Start node is out of the bounds of the graph\n";
    return 1;
  }
  GNode start = *std::next(graph.begin(), startNode);
  deltaStepAlgo<UpdateRequest>(graph, start, ReqPushWrap(), OutEdgeRangeFn{graph});

  return 0;
}
```

Пример запуска:





PageRank

PageRank — один из алгоритмов ссылочного ранжирования. Алгоритм применяется к коллекции документов, связанных гиперссылками, и назначает каждому из них некоторое численное значение, измеряющее его «важность» или «авторитетность» среди остальных документов. Алгоритм может применяться не только к веб-страницам, но и к любому набору объектов, связанных между собой взаимными ссылками, то есть к любому графу.

Код параллельной реализации с использованием Galois:

```
#include "galois/Bag.h"
#include "galois/Galois.h"
#include "galois/Timer.h"
#include "galois/graphs/LCGraph.h"
#include "galois/graphs/TypeTraits.h"

constexpr static const float ALPHA        = 0.85;
constexpr static const float INIT_RESIDUAL = 1 - ALPHA;
constexpr static const float tolerance   = 1.0e-3;
constexpr static const unsigned maxIterations = 1000;
constexpr static const unsigned numThreads = 4;
constexpr static const bool skipVerify = false;
constexpr static const unsigned CHUNK_SIZE = 16;

static inline float atomicAdd(std::atomic<float>& v, float delta) {
  float old;
  do {
    old = v;
  } while (!v.compare_exchange_strong(old, old + delta));
  return old;
}
```

```
template <typename GNode>
struct TopPair {
  float value;
  GNode id;

  TopPair(float v, GNode i) : value(v), id(i) {}

  bool operator<(const TopPair& b) const {
    if (value == b.value)
      return id > b.id;
    return value < b.value;
  }
};

template <typename Graph>
void printTop(Graph& graph, unsigned topn = 20) {

  using GNode = typename Graph::GraphNode;
  typedef TopPair<GNode> Pair;
  typedef std::map<Pair, GNode> TopMap;

  TopMap top;

  for (auto ii = graph.begin(), ei = graph.end(); ii != ei; ++ii) {
    GNode src  = *ii;
    auto& n    = graph.getData(src);
    float value = n.value;
    Pair key(value, src);

    if (top.size() < topn) {
      top.insert(std::make_pair(key, src));
      continue;
    }

    if (top.begin()->first < key) {
      top.erase(top.begin());
      top.insert(std::make_pair(key, src));
    }
  }

  int rank = 1;
  std::cout << "Rank PageRank Id\n";
  for (auto ii = top.rbegin(), ei = top.rend(); ii != ei; ++ii, ++rank) {
    std::cout << rank << ": " << ii->first.value << " " << ii->first.id << "\n";
```

```
  }
}

struct LNode {
  float value;
  std::atomic<float> residual;

  LNode() {
    value = 0.0;
    residual = INIT_RESIDUAL;
  }

  friend std::ostream &operator<<(std::ostream &os, const LNode &n) {
    os << "{PR " << n.value << ", residual " << n.residual << "}";
    return os;
  }
};

typedef galois::graphs::LC_CSR_Graph<LNode, void>::with_numa_alloc<
    true>::type ::with_no_lockable<true>::type Graph;
typedef typename Graph::GraphNode GNode;

void asyncPageRank(Graph &graph) {
    typedef galois::worklists::PerSocketChunkFIFO<CHUNK_SIZE> WL;
    galois::for_each(
        galois::iterate(graph),
        [&](GNode src, auto &ctx) {
            LNode &sdata = graph.getData(src);
            constexpr const galois::MethodFlag flag =
                galois::MethodFlag::UNPROTECTED;

            if (sdata.residual > tolerance) {
                float oldResidual = sdata.residual.exchange(0.0);
                sdata.value += oldResidual;
                int src_nout = std::distance(graph.edge_begin(src, flag),
                                             graph.edge_end(src, flag));
                if (src_nout > 0) {
                    float delta = oldResidual * ALPHA / src_nout;
                    for (auto jj : graph.edges(src, flag)) {
                        GNode dst = graph.getEdgeDst(jj);
                        LNode &ddata = graph.getData(dst, flag);
                        if (delta > 0) {
                            auto old = atomicAdd(ddata.residual, delta);
                            if ((old < tolerance) && (old + delta >= tolerance)) {
```

```
                        ctx.push(dst);
                    }
                }
            }
        }
    },
    galois::loopname("PushResidualAsync"),
    galois::disable_conflict_detection(), galois::no_stats(),
    galois::wl<WL>());
}

int main(int argc, char **argv) {
  galois::SharedMemSys G;

  if (argc < 2) {
    std::cerr << "Usage: " << argv[0] << " <input graph file>\n";
    return 1;
  }

  std::string inputFile = argv[1];

  Graph graph;
  galois::graphs::readGraph(graph, inputFile);
  std::cout << "Read " << graph.size() << " nodes, " << graph.sizeEdges()
            << " edges\n";

  galois::preAlloc(5 * numThreads +
              (5 * graph.size() * sizeof(typename Graph::node_data_type)) /
                 galois::runtime::pagePoolSize());

  std::cout << "tolerance:" << tolerance << ", maxIterations:" << maxIterations
            << "\n";

  galois::do_all(
    galois::iterate(graph), [&graph](GNode n) { graph.getData(n); },
    galois::no_stats(), galois::loopname("Initialize"));

  galois::StatTimer execTime("Timer_0");
  execTime.start();

  asyncPageRank(graph);

  execTime.stop();
```

```
    if (!skipVerify) {
        printTop(graph);
    }

    return 0;
}
```

Пример запуска:

Для запуска возьмем граф CA-AstroPh.

Результат: