УДК 519.688

РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АЛГОРИТМОВ РЕШЕНИЯ НЕКОТОРЫХ ОПТИМИЗАЦИОННЫХ ЗАДАЧ НА ГРАФАХ *

А. М. Сазонов А. В. Соколов С. А. Старков

ИПМИ Карельского научного центра РАН,
Петрозаводский государственный университет

email: sazonov@cs.karelia.ru email: sokavs@gmail.com email: starkov@cs.karelia.ru

Аннотация. В статье дано описание реализованной на языке C++ с использованием технологии параллельного программирования OpenMP библиотеки параллельных алгоритмов решения некоторых оптимизационных задач на графах. Библиотека включает параллельные алгоритмы Флойда, Джонсона и Краскала. Также реализованы функции ввода, вывода, добавления и удаления вершин и рёбер, и последовательные реализации алгоритмов Дейкстры и Беллмана-Форда. Предусмотрена работа с 3 типами представления графа: список смежности, матрица смежности, список рёбер. Приводится сравнение реализованных в библиотеке алгоритмов с реализацией этих алгоритмов в широко используемой в мире библиотеке BGL.

Исходя из результатов экспериментов, можно сделать вывод, что применение параллельного алгоритма Флойда на полных графах целесообразно. Однако максимальное ускорение можно получить на графах с количеством вершин не менее 1000. Для графов с меньшим количеством вершин использование параллельного алгоритма Флойда не позволяет получить максимальное ускорение, так как в этом случае существенны накладные расходы. Также можно выделить тенденцию, что с возрастанием количества вершин графа наибольшее ускорение получается при использовании наибольшего количества потоков. Напротив, для небольших графов выгоднее использовать меньшее количество потоков.

Применение параллельного алгоритма Джонсона позволяет получить только слабое ускорение. Следует отметить, что для полных графов ускорение незначительно лучше. Применение параллельного алгоритма Краскала даёт ускорение на графах с большим числом рёбер. На небольших графах ускорение несущественно.

Ключевые слова и фразы: параллельный алгоритм Флойда; параллельный алгоритм Джонсона; параллельный алгоритм Краскала.

1 ВВЕДЕНИЕ

Поиск кратчайших путей в графе и построение минимального остов-

Работа выполнена при финансовой поддержке Программы стратегического развития ПетрГУ в рамках реализации комплекса мероприятий по развитию научно-исследовательской деятельности.

ного дерева – важные задачи, возникающие в разных областях науки и техники. В настоящее время наиболее распространённой библиотекой, использующейся для решения этих задач, является BGL (Boost Graph Library) [4]. Также для работы с графами используется библиотека LEDA [5]. Однако эти библиотеки включают в себя *только последовательные* версии алгоритмов для решения задач на графах. В данной статье мы представляем параллельные реализации некоторых полиномиальных алгоритмов на графах. В будущем планируется разработать параллельные версии решения некоторых NP-задач, таких, например, как задача Штейнера [6] или задача коммивояжёра [7,8].

2 ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ФЛОЙДА

Алгоритм Флойда предназначен для поиска кратчайших путей между всеми парами вершин графа.

Параллелизм, применимый для алгоритма Флойда – итерационный. Эффективный способ распараллеливания данного алгоритма состоит в одновременном обновлении значений матрицы смежности, т.е. для каждой промежуточной вершины k вычислять путь из произвольной вершины i в произвольную вершину j параллельно. Данный способ параллельности корректен и не приведет к коллизиям.

Доказательство корректности. Необходимо доказать, что на каждой итерации внешнего цикла обновление матрицы смежности может выполняться независимо. Т.е. необходимо показать, что на итерации k не изменяются значения A[i][k], A[k][j] для любых i, j, так как все вычисления на данной итерации зависят только от этих величин.

A[i][j] = min(A[i][j], A[i][k] + A[k][j]):

- для i = k: A[i][k] = min(A[i][k], A[i][k] + A[k][k]). A[k][k] = 0 => A[i][k] не изменится.
- для j = k: A[k][j] = min(A[k][j], A[k][k] + A[k][j]). A[k][k] = 0 => A[k][j] не изменится.

Приведём псевдокод для параллельного алгоритма Флойда. Пусть граф задан матрицей смежности A[n][n]. Функция min, возвращающая минимум из 2 чисел, должна учитывать способ указания в матрице смежности несуществующих дуг графа. В данной реализации длина несуществующих дуг полагается равной бесконечности.

```
void Floyd (A)
  for (k = 0; k < n; k++)
    parallel_for (i = 0; i < n; i++)
      if (A[i][k] ≠ ∞)
        for (j = 0; j < n; j++)
          if (A[k][j] ≠ ∞)
            A[i][j] = min(A[i][j], A[i][k] + A[k][j]);
```

(Здесь `parallel_for` – параллельный цикл.)

Сложность последовательной версии данного алгоритма – $O(n^3)$, параллельной – $O(n^3/p)$, где p – число процессоров. Алгоритм Флойда работает с матрицей смежности, следовательно его эффективнее применять для графов, близких к полным.

3 ПАРАЛЛЕЛЬНЫЙ АЛГОРИТМ ДЖОНСОНА

Алгоритм Джонсона предназначен для поиска кратчайших путей между всеми парами вершин графа. Идея алгоритма Джонсона – применение алгоритма Дейкстры к каждой вершине. Однако в связи с этим возникает проблема, так как алгоритм Дейкстры применим только для графов без рёбер отрицательного веса. Чтобы преобразовать граф с отрицательными весами рёбер, не содержащий циклов отрицательного веса, к графу с неотрицательными весами рёбер применяется алгоритм Беллмана-Форда.

Преобразование исходного графа к графу без рёбер отрицательного веса занимает незначительное время от общего времени работы алгоритма, поэтому его распараллеливание нецелесообразно. Эффективный способ распараллеливания состоит в одновременном применении алгоритма Дейкстры для каждой вершины. Алгоритм Дейкстры может работать с разными вершинами параллельно, так как граф не изменяется во времени и массивы расстояний и предшествования независимы.

Пусть дан граф G = <V, E>. Вводится весовая функция: w'(u,v) = w(u,v) + h(u) – h(v). h: V → R – некая числовая функция от вершины (вес вершины). Далее находятся кратчайшие пути в графе с весами w' и происходит возврат к исходному графу.

Для построения графа G' = <V', E'> с весами w' и поиска функции h применима следующая процедура:

V' = V ∪ {s} ;

E' = E ∪ {(s,v) для всех v} ;

w(s,v) = 0 для всех v .

В полученном графе с помощью алгоритма Беллмана-Форда находятся кратчайшие пути из вершины s во все остальные.

Далее приведём псевдокод *параллельного* алгоритма Джонсона. Пусть G – обрабатываемый граф, s – добавляемая вершина для построения графа G' (без рёбер отрицательного веса). После выполнения алгоритма массив d будет содержать длины кратчайших путей между всеми парами вершин, а массив path – сами пути.

```
bool Johnson (G, s, d, path)
  Построить G'
```

```
f = BellmanFord (G', s, h)
if (f = false) then
  print('Граф содержит циклы с отрицательным весом')
  return false
for (u,v) ∈ E do
  w(u,v) = w(u,v) + h(u) – h(v)
parallel_for u ∈ V do
  Dijkstra (G, u, d[u], path[u])
for (u,v) ∈ E do
  d(u,v) = d(u,v) – h(u) + h(v)
return true
```

Сложность последовательной версии данного алгоритма – O(V(VlogV + ElogV) + VE), тогда как сложность алгоритма Беллмана-Форда – O(VE), а алгоритма Дейкстры – O(VlogV + ElogV), и при этом алгоритм Дейкстры нужно применить к каждой вершине. Сложность параллельной версии алгоритма Джонсона равна O(V(VlogV + ElogV)/p + VE), где p – число процессоров. Данный алгоритм работает со списком смежности, поэтому он эффективнее для разрежённых графов.

4 ОБЪЕКТ-ГРАФ

Класс Graph содержит:

- поля:
  - `AdjMatrix`
  - `AdjList`
  - `EdgeList`
  - `currentType`
  - `infinity`
  - `directed`
- методы:
  - `input()`
  - `output()`
  - `resize()`
  - `verticesCount()`
  - `addVertex()`
  - `addEdge()`
  - `deleteEdge()`
  - `isAdjMatrix()`
  - `isAdjList()`
  - `isEdgeList()`
  - `toAdjMatrix()`
  - `toAdjList()`
  - `toEdgeList()`
  - `AdjMatrixToAdjList()`
  - `AdjListToAdjMatrix()`
  - `AdjMatrixToEdgeList()`

- `AdjListToEdgeList()`
- `EdgeListToAdjMatrix()`
- `EdgeListToAdjList()`

Поле AdjMatrix служит для хранения графа в виде матрицы смежности, AdjList – в виде списка смежности, EdgeList – в виде списка рёбер. Поле currentType указывает текущий тип представления графа. Методы input() и output() предназначены для ввода и вывода графа. Поле infinity служит для представления бесконечности для используемого графа. Бесконечность служит для обозначения длин несуществующих рёбер. Поле directed служит для определения, является ли граф направленным.

Метод addVertex() предназначен для добавления вершины, addEdge() и deleteEdge() для добавления и удаления ребра, resize() для изменения числа вершин графа, verticesCount() для подсчёта числа вершин.

В данной библиотеке рассматриваются такие виды представления графа в памяти как матрица смежности, список смежности и список рёбер. Различные алгоритмы предназначены для работы с разными типами представления графа. Например, алгоритм Флойда работает с матрицей смежности, а алгоритм Джонсона со списком смежности. Для обеспечения поддержки работы с несколькими способами хранения графа, класс Graph содержит поля для его представления: AdjMatrix, AdjList, EdgeList, соответственно для хранения матрицы смежности, списка смежности и списка рёбер. Для преобразования графа из одного типа представления в другой используются методы AdjMatrixToAdjList(), AdjListToAdjMatrix(), EdgeListToAdjList(), AdjMatrixToEdgeList(), AdjListToEdgeList(), EdgeListToAdjMatrix(). Данные методы не доступны пользователю и используются для реализации методов toAdjList(), toAdjMatrix(), toEdgeList(). Методы isAdjMatrix(), isAdjList(), isEdgeList() предназначены для получения информации о том, в каком виде хранится граф. Методы toAdjList(), toAdjMatrix(), toEdgeList() доступны пользователю, и позволяют конвертировать граф в нужный тип представления, из любого другого.

5 КЛАСС ДЛЯ ХРАНЕНИЯ РЕБРА ГРАФА

Класс Edge для хранения ребра графа содержит:

- поля:
  - `u : unsigned`
  - `v : unsigned`
  - `weight : int`
- методы:
  - `operator>>`

- `operator<<`
- `operator<`

Поля u и v хранят соответственно начало и конец ребра, поле weight служит для хранения веса ребра. Перегруженные операторы >>, << осуществляет ввод и вывод ребра в формате <начало ребра> <конец ребра> <вес ребра>, где начало и конец ребра – целые неотрицательные числа, вес ребра – целое число. Перегруженный оператор < служит для сравнения рёбер по весу.

6 КЛАСС ДЛЯ ХРАНЕНИЯ СПИСКА СМЕЖНОСТИ

Класс AdjList для хранения списка смежности графа содержит:

- поля:
  - `G : vector < vector < pair < int, unsigned > > >`
  - `maxVertexNum : int`
  - `infinity : int`
  - `directed : bool`
- методы:
  - `operator>>`
  - `operator<<`
  - `operator []`
  - `resize()`
  - `addVertex()`
  - `addEdge()`
  - `deleteEdge()`
  - `toAdjMatrix()`
  - `toEdgeList()`

Поле G служит для хранения графа в виде списка смежности. Поле maxVertexNum – максимальный номер вершины, входящей в граф. Поле infinity служит для представления бесконечности для используемого графа. Поле directed служит для определения, является ли граф направленным. Методы toAdjMatrix(), toEdgeList() служат для преобразования списка смежности соответственно к матрице смежности и списку рёбер. Методы, общие для всех классов представления графа, описаны далее в разделе «Общие методы для классов AdjList, AdjMatrix, EdgeList».

7 КЛАСС ДЛЯ ХРАНЕНИЯ МАТРИЦЫ СМЕЖНОСТИ

Класс AdjMatrix для хранения списка смежности графа содержит:

- поля:
  - `G : vector <vector<int>>`
  - `maxVertexNum : int`
  - `infinity : int`
  - `directed : bool`
- методы:

- operator>>
- operator<<
- operator []
- resize()
- addVertex()
- addEdge()
- deleteEdge()
- toAdjList()
- toEdgeList()

Поле G служит для хранения графа в виде матрицы смежности. Поле maxVertexNum – максимальный номер вершины, входящей в граф. Поле infinity служит для представления бесконечности для используемого графа. Поле directed служит для определения, является ли граф направленным. Методы toAdjList(), toEdgeList() служат для преобразования матрицы смежности соответственно к списку смежности и списку рёбер. Методы, общие для всех классов представления графа, описаны далее в разделе «Общие методы для классов AdjList, AdjMatrix, EdgeList».

8 КЛАСС ДЛЯ ХРАНЕНИЯ СПИСКА РЁБЕР

Класс EdgeList для хранения списка смежности графа содержит:

- поля:
  - G : vector < Edge >
  - maxVertexNum : int
  - infinity : int
  - directed : bool
- методы:
  - operator>>
  - operator<<
  - operator []
  - resize()
  - size()
  - addVertex()
  - addEdge()
  - deleteEdge()
  - toAdjList()
  - toAdjMatrix()
  - sort()
  - cost()

Поле G служит для хранения графа в виде списка смежности. Поле maxVertexNum – максимальный номер вершины, входящей в граф. Поле infinity служит для представления бесконечности для используемого графа. Поле directed служит для определения, является ли граф направленным. Методы toAdjList(), toAdjMatrix() служат для преобра-

зования списка рёбер соответственно к списку смежности и матрице смежности. Метод size() возвращает количество рёбер в графе. Методы, общие для всех классов представления графа, описаны далее в разделе «Общие методы для классов AdjList, AdjMatrix, EdgeList».

9 ОБЩИЕ МЕТОДЫ КЛАССОВ AdjList, AdjMatrix, EdgeList

`operator>>`

- Описание: метод осуществляет заполнение структуры, содержащей информацию о графе, со стандартного ввода. Граф вводится в формате <начало ребра>, <конец ребра>, <вес ребра>, где начало и конец ребра – целые неотрицательные числа, вес ребра – целое число. В случае несовпадения формата ввода с указанным генерируется исключение “Wrong input format” и метод возвращает значение 1. Список смежности AdjList заполняется следующим образом: для каждой вершины записываются все ее соседи и веса рёбер до соседей в виде пар <вес ребра, соседняя вершина>. Матрица смежности AdjMatrix заполняется весами рёбер для каждой пары вершин. Список рёбер EdgeList заполняется следующим образом: для каждого ребра записываются его начало, конец и вес.
- Возвращаемое значение:
  - 0, в случае успешного ввода
  - 1, иначе

`operator<<`

- Описание: метод осуществляет вывод информации о графе в стандартный поток вывода. Граф выводится в формате, описанном выше (см. operator>>).

`operator [](int i)`

- Описание: метод возвращает i-й элемент структуры для каждого типа представления графа.
- Аргументы:
  - i: int – номер элемента
- Возвращаемое значение: i-й элемент структуры для представления графа.

`resize(unsigned size)`

- Описание: метод изменяет число вершин графа на значение size.
- Возвращаемое значение: нет.

`addEdge(unsigned FirstVertexNum,`
`unsigned SecondVertexNum, int Weight)`

- Описание: метод добавляет в структуру, содержащую инфор-

мацию о графе, ребро с указанными вершинами и весом. В случае существования в графе добавляемого ребра генерируется исключение “This edge exists”, и метод возвращает 1.

- Аргументы:
  - FirstVertexNum:unsigned int номер первой вершины добавляемого ребра
  - SecondVertexNum:unsigned int номер второй вершины добавляемого ребра
  - Weight:int вес добавляемого ребра
- Возвращаемое значение:
  - 0, в случае успешного добавления ребра
  - 1, иначе

`addVertex()`

- Описание: метод добавляет в структуру, содержащую информацию о графе из N вершин, вершину с номером N + 1.
- Аргументы: нет
- Возвращаемое значение: нет

```
deleteEdge(unsigned FirstVertexNum,
unsigned SecondVertexNum)
```

- Описание: метод удаляет из структуры, содержащей информацию о графе ребро с указанными вершинами. В случае отсутствия в графе удаляемого ребра генерируется исключение "This edge not exists" и метод возвращает значение 1.
- Аргументы:
  - FirstVertexNum:unsigned int номер первой вершины добавляемого ребра
  - SecondVertexNum:unsigned int номер второй вершины добавляемого ребра
- Возвращаемое значение:
  - 0, в случае успешного удаления ребра
  - 1, иначе

10 ФУНКЦИИ БИБЛИОТЕКИ, РЕАЛИЗУЮЩИЕ АЛГОРИТМЫ

```
Dijkstra(Graph &g, vector<int> &distance,
vector<unsigned> &predecessor, unsigned start)
```

- Описание: функция осуществляет поиск кратчайших путей от заданной вершины до остальных методом Дейкстры. В случае, если исходной вершины не существует, генерируется исключение "Start vertex not exists" и метод возвращает значение 1. Для применения данной функции граф не должен содержать рёбер отрицательного

веса, в противном случае генерируется исключение "Negative edges are unacceptable" и метод возвращает значение 2.

- Аргументы:
  - g: Graph – исходный граф
  - start: unsigned int – номер исходной вершины
  - distance:vector<int> – массив, который будет заполнен длинами кратчайших путей от заданной вершины до остальных (массив расстояний)
  - predecessor:vector<unsigned int> – массив, который будет заполнен кратчайшими путями от заданной вершины до остальных (массив предшествования)
- Возвращаемое значение:
  - 0, в случае успешного выполнения
  - 1, если указанная исходная вершина не входит в граф
  - 2, если граф содержит ребра отрицательного веса

```
BellmanFord(Graph &g, vector<int> &distance,
unsigned start)
```

- Описание: функция осуществляет поиск кратчайших путей от заданной вершины до остальных методом Беллмана-Форда.
- Аргументы:
  - g:Graph – исходный граф
  - start: unsigned int – номер исходной вершины
  - distance: vector<int> – массив, который будет заполнен длинами кратчайших путей от заданной вершины до остальных (массив расстояний)
- Возвращаемое значение:
  - true, если граф содержит цикл отрицательного веса
  - false, иначе

```
ParallelJohnson(Graph &g,
vector <vector<int>> &distance)
```

- Описание: функция осуществляет поиск кратчайших путей между всеми парами вершин методом Джонсона. В результате выполнения заполняется матрица кратчайших путей distance. Функция поддерживает параллельную работу на нескольких процессорах многопроцессорной системы.
- Аргументы:
  - g:Graph – исходный граф

- distance:vector <vector<int>> – матрица, которая будет заполнена длинами кратчайших путей между всеми парами вершин.

- Возвращаемое значение: нет.

```
ParallelFloyd(Graph &g,
vector <vector<int>> &distance)
```

- Описание: функция осуществляет поиск кратчайших путей между всеми парами вершин методом Флойда. В результате выполнения заполняется матрица кратчайших путей distance. Функция поддерживает параллельную работу на нескольких процессорах многопроцессорной системы.
- Аргументы:
  - g:Graph – исходный граф
  - distance:vector <vector<int>> – матрица, которая будет заполнена длинами кратчайших путей между всеми парами вершин.
- Возвращаемое значение: нет.

```
ParallelKruskal(Graph &g, vector<Edge> &mst)
```

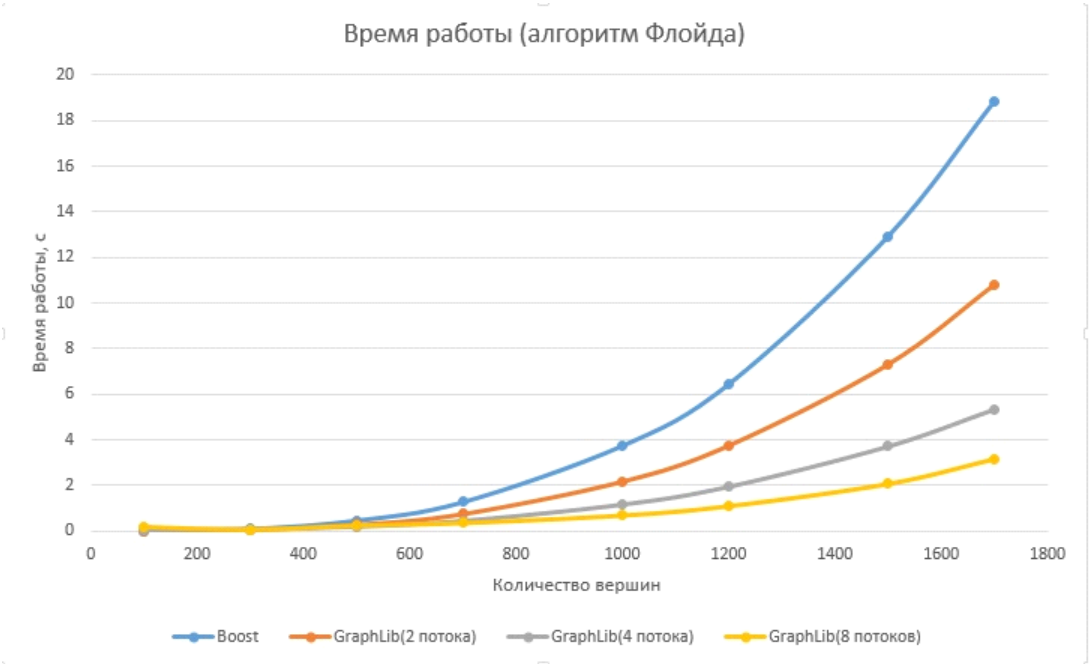
- Описание: функция осуществляет поиск минимального остовного дерева алгоритмом Краскала. Функция поддерживает параллельную работу на нескольких процессорах многопроцессорной системы.
- Аргументы:
  - g: Graph – исходный граф
  - mst:vector <Edge> – список рёбер, включенных в минимальный остов.
- Возвращаемое значение: нет.

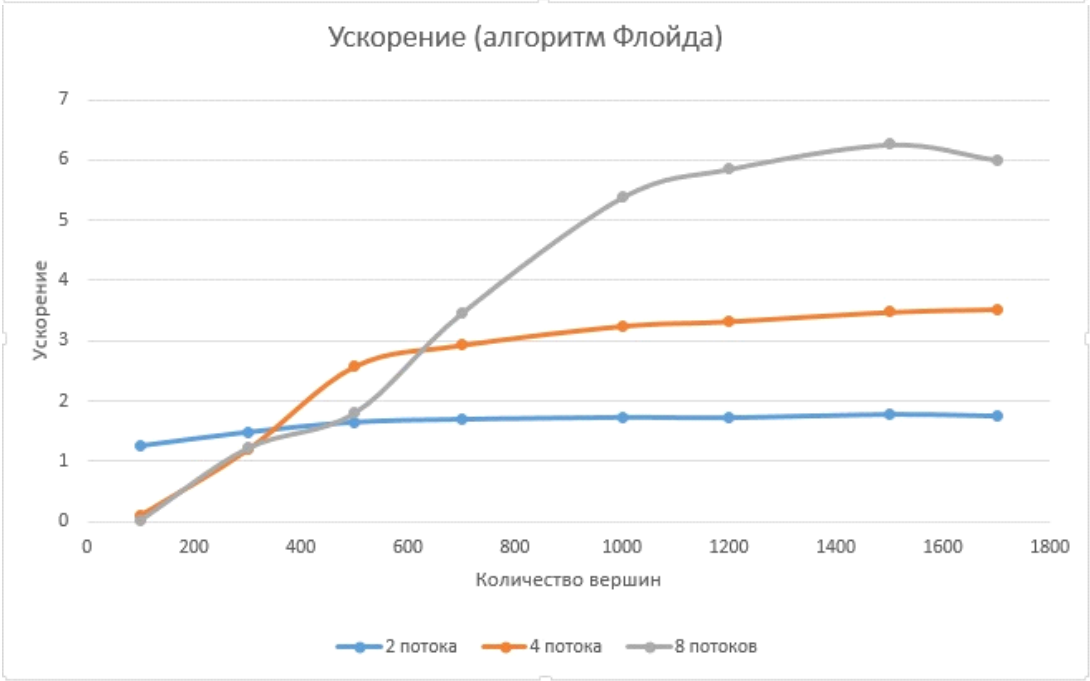
11 РЕЗУЛЬТАТЫ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

Вычислительные эксперименты включали в себя сравнение производительности параллельных алгоритмов разработанной библиотеки с последовательными аналогами, реализованными в библиотеке BGL. Данный выбор не является идеальным, так как библиотека BGL не содержит параллельных версий сравниваемых алгоритмов. Однако в процессе поиска библиотек для сравнения не было найдено библиотеки, содержащей параллельные версии необходимых алгоритмов. Библиотека BGL была выбрана, так как в настоящее время она получила наиболее широкое распространение для задач, связанных с обработкой графов, и версия данной библиотеки установлена на кластере КарНЦ РАН [9].

Таким образом, основным вопросом исследования стало ускорение, которое позволяет получить соответствующий параллельный алгоритм по сравнению с последовательным аналогом, то есть отношение времени работы последовательного алгоритма к времени работы параллельного. Сравнение проводилось на кластере КарНЦ РАН с использованием 2, 4 и 8 потоков для обработки.

1. Параллельный алгоритм Флойда.

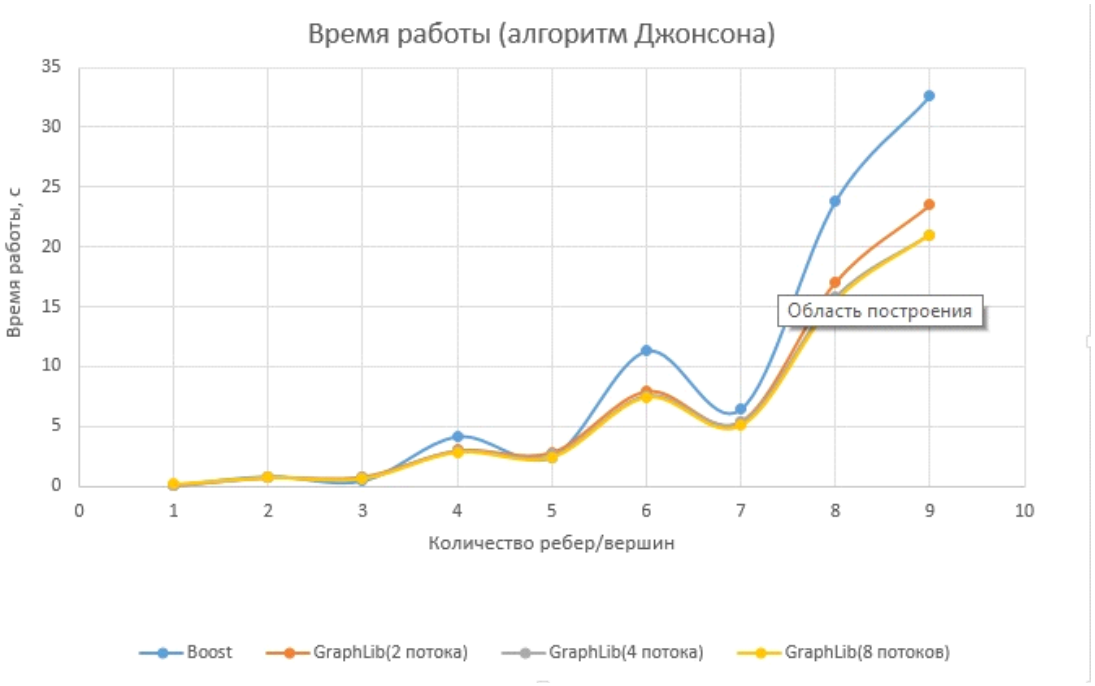


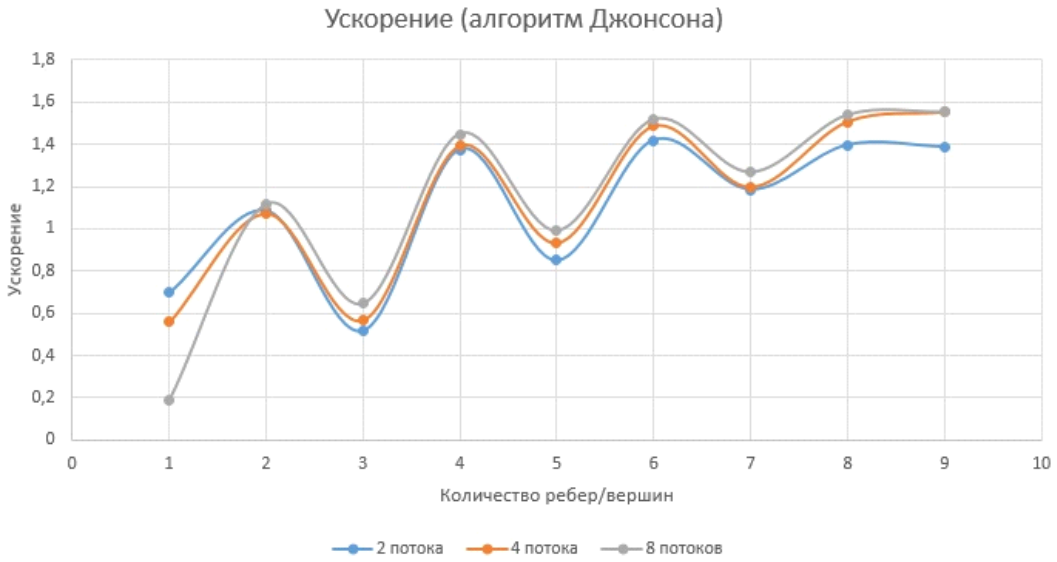


Исходя из результатов экспериментов, можно сделать вывод, что применение параллельного алгоритма Флойда на полных графах целе-

сообразно. Однако максимальное ускорение можно получить на графах с количеством вершин не менее 1000. Для графов с меньшим количеством вершин использование параллельного алгоритма Флойда не позволяет получить максимальное ускорение, так как в этом случае существенны накладные расходы. Также можно выделить тенденцию, что с возрастанием количества вершин графа наибольшее ускорение получается при использовании наибольшего количества потоков. Напротив, для небольших графов выгоднее использовать меньшее количество потоков.

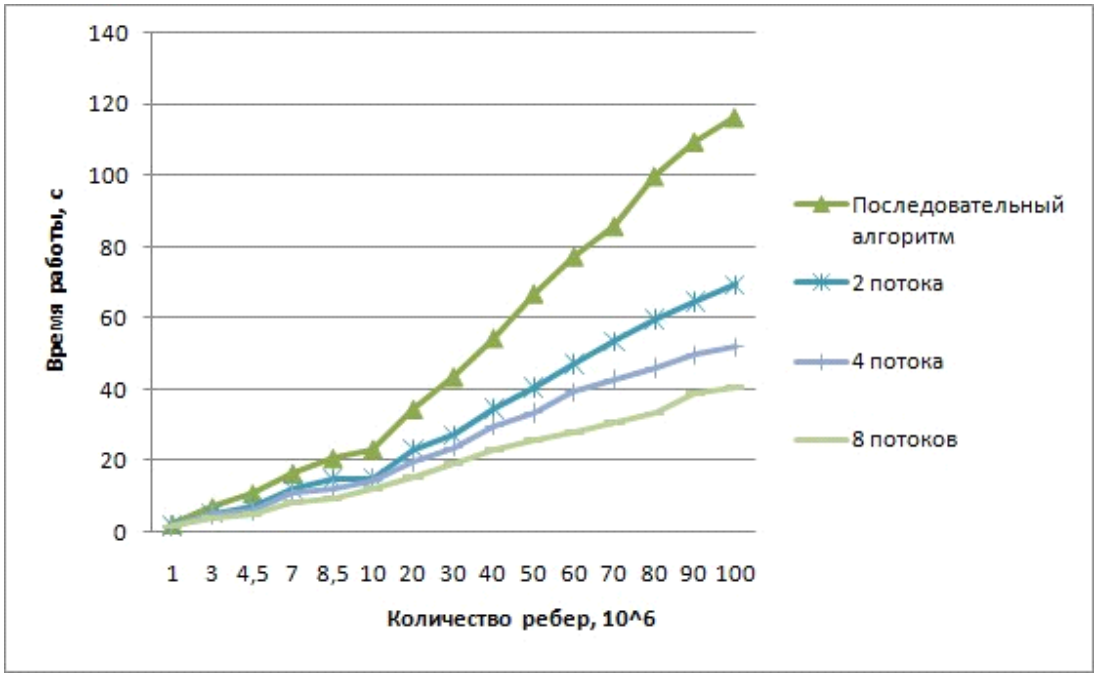
2. Параллельный алгоритм Джонсона

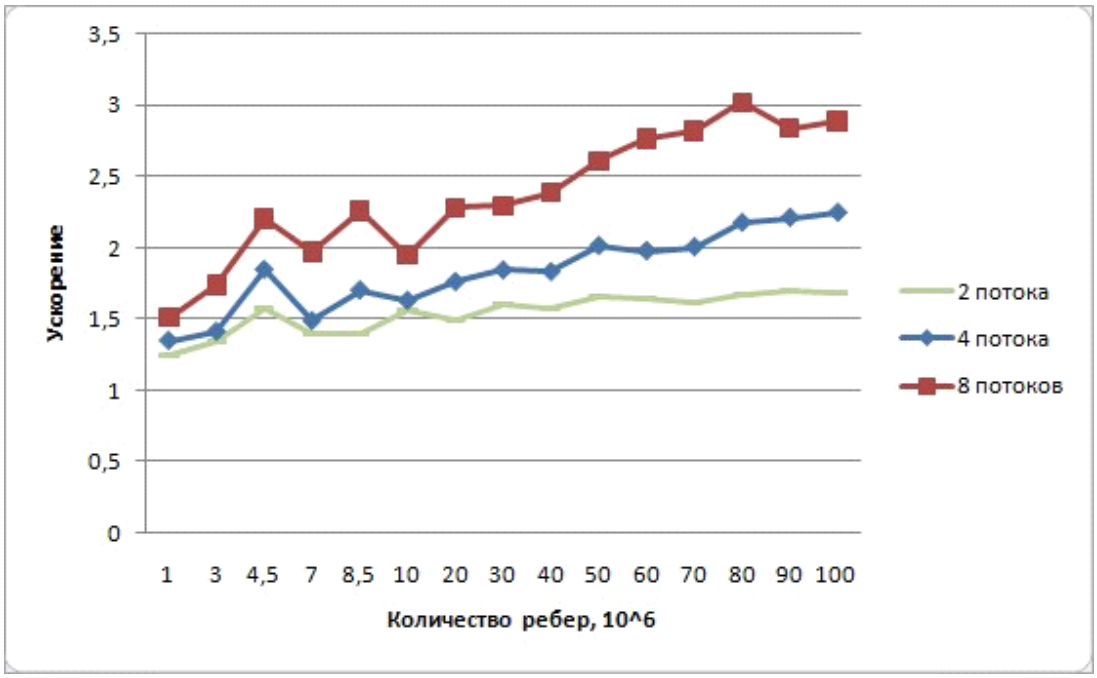




Исходя из результатов экспериментов, можно сделать вывод, что применение параллельного алгоритма Джонсона позволяет получить только слабое ускорение. Следует отметить, что для полных графов ускорение незначительно лучше.

3. Параллельный алгоритм Краскала





Исходя из результатов экспериментов, можно сделать вывод, что применение параллельного алгоритма Краскала даёт ускорение на

графах с большим числом рёбер. На небольших графах ускорение несущественно.

12 ЗАКЛЮЧЕНИЕ

В результате работы была разработана небольшая библиотека для решения некоторых оптимизационных задач на графах, включающая в себя параллельные алгоритмы Флойда, Джонсона и Краскала. Вычислительные эксперименты, состоящие в сравнении времени работы данных параллельных алгоритмов разработанной библиотеки с последовательными аналогами, реализованными в библиотеке BGL, показали, что параллельный алгоритм Флойда позволяет получить значительно лучшее ускорение, чем параллельный алгоритм Джонсона, а параллельный алгоритм Краскала эффективен при большом количестве рёбер.

СПИСОК ЛИТЕРАТУРЫ

1. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. (2005), *Алгоритмы: построение и анализ, 2-е издание*, Вильямс, М., 1296 с.
2. Гергель, В. П. (2010), *Высокопроизводительные вычисления для многоядерных систем*, Изд-во ННГУ им. Н. И. Лобачевского, Нижний Новгород, 421 с.
3. Антонов, А. С. (2009), *Параллельное программирование с использованием технологии OpenMP: уч. пособие*, Изд-во МГУ, М, 77 с.
4. Boost C++ Libraries [Электронный ресурс], Режим доступа: http://www.boost.org/
5. LEDA: Combinatorial and Graph Algorithms [Электронный ресурс], Режим доступа: http://www.comp.nus.edu.sg/~cs5234/LEDA-Info/leda-info.htm
6. Иванов, А. О., Тужилин, А. А. (1991), «Задача Штейнера на плоскости или плоские минимальные сети», *Матем. сб.*, Т. 182. № 12, сс. 1813–1844.
7. Громкович, Ю. (2014), «Детерминированные подходы к алгоритмизации труднорешаемых задач. Часть I: Основные понятия», *Эвристические алгоритмы и распределенные вычисления*. Т. 1, № 2, сс. 30–42.
8. Макаркин, С. Б., Мельников, Б. Ф. (2013), «Геометрические методы решения псевдогеометрической версии задачи коммивояжера», *Стохастическая оптимизация в информатике*, Т. 9, вып. 2, сс. 54–72.
9. ЦКП Карельского научного центра РАН [Электронный ресурс], Режим доступа: http://cluster.krc.karelia.ru/section.php?id=2

REALIZATION OF SOME PARALLEL ALGORITHMS OF SOLVING SOME OPTIMIZATION PROBLEMS ON GRAPHS

A. M. Sazonov A. V. Sokolov S. A. Starkov
Institute of Applied Mathematical Research
of the Karelian Research Centre of the Russian Academy of Sciences,
Petrozavodsk State University
email: sazonov@cs.karelia.ru email: sokavs@gmail.com email: starkov@cs.karelia.ru

Abstract. This article describes the library of parallel graph algorithms with using OpenMP technology implemented in C++. The library includes some parallel algorithms such as Floyd, Johnson and Kruskal. Additionally implemented methods of input, output, adding and removing vertices and edges, consecutive algorithms of Dijkstra and Bellman-Ford. This library supports 3 types of graph representation: adjacency list, adjacency matrix, list of edges. The article provides performance comparison between developed library and widely known BGL.

Based on the experimental results, we can conclude that the application of parallel Floyd algorithm on the complete graph is advisable. However, the maximum acceleration can only be obtained on a graph with the number of vertices greater than 1000. For graphs with fewer vertices usage of the parallel Floyd algorithm does not yield the maximum acceleration, as in this case, there are significant overhead costs. It is also possible to note a tendency – with increasing number of vertices the largest acceleration is obtained using the maximum number of threads. Conversely, for smaller graphs it is advantageous to use a smaller number of threads. The use of parallel Johnson algorithm allows to get only a slight acceleration. It should be noted that for complete graphs acceleration is marginally better. The use of parallel algorithm Kruskal gives acceleration to graphs with a large number of edges. On smaller graphs acceleration is insignificant.

Keywords and phrases: parallel Floyd's algorithm; parallel Johnson's algorithm; parallel Kruskal's algorithm.

REFERENCES

1. Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009), *Introduction to Algorithms, 3rd edition*, MIT Press and McGraw-Hill, Massachusetts, 1312 p.
2. Gergel, V. P. (2010), *High-performance computing for multi-core systems [Vysokoproizvoditel'nye vychisleniya dlya mnogoyadernyh sistem]*, Nizhniy Novgorod State Univ. Ed., Nizhniy Novgorod, 421 p.
3. Antonov, A. S. (2009), *Parallel programming with using OpenMP technology: tutorial [Parallel'noe programmirovanie s ispol'zovaniem tehnologii OpenMP: uchebnoe posobie]*, Moscow State Univ. Ed., Moscow, 77 p.
4. Boost C++ Libraries [Electronic resource], Access mode: http://www.boost.org/
5. LEDA: Combinatorial and Graph Algorithms [Electronic resource], Access mode: http://www.comp.nus.edu.sg/~cs5234/LEDA-Info/leda-info.htm
6. Ivanov, A. O., Tuzhilin, A. A. (1991), "The Steiner problem in the plane or plane minimal nets", *Math. collected papers* ["Zadacha Shteynera na ploskosti ili ploskie minimal;nye seti", *Matematicheskiy sbornik*], Vol. 182, No. 12, pp. 1813–1844.
7. Hromkovič, J. (2014) "Deterministic approaches to algorithmic for hard computing problems. Part I: The main definitions", *Heuristic algorithms and distributed computing* ["Determinirovannye podhody k algoritmizacii trudnoreshaemyh

zadach. Chast' I: Osnovnye opredeleniya", *Evristicheskie algoritmy i raspredelyonnye vychisleniya*], Vol. 1, No. 2, pp. 30–42, Access mode: http://samsu.ru/ru/node/4736

8. Makarkin, S. B., Melnikov, B. F. (2013), "Geometrical methods for solving the pseudogeometrical version of the travelling salesman problem", *Stochastic optimization in computer science* ["Geometricheskie metody resheniya psevdogeometricheskoy versii zadachi kommivoyazhera", *Stohasticheskaya optimizacita v informatike*], Vol. 9, No. 2, pp. 54–72.
9. Center for collective use of Karelian Research Centre of Russian Academy of Sciences [Electronic resource], Access mode: http://cluster.krc.karelia.ru/section.php?id=2