МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Нижегородский государственный университет им. Н.И. Лобачевского

А.Ю. Пирова

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РАЗДЕЛЕНИЯ ГРАФОВ

Учебно-методическое пособие

Рекомендовано методической комиссией ИИТММ для студентов ННГУ,
обучающихся по направлению подготовки

01.03.02 Прикладная математика и информатика

Нижний Новгород
2019

УДК 519.688
ББК 32.973
П-33

П-33 Пирова А.Ю. Параллельные алгоритмы разделения графов: учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2019. – 20 с.

Рецензент: к.т.н., доцент Карпенко С.Н.

В учебном пособии рассматривается задача разделения графов. Приводится обзор методов разделения и подробное описание последовательного и параллельного алгоритма многоуровневого разделения. Дается пример использования реализации алгоритма разделения графа из библиотеки ParMETIS.

Практикум предназначен для студентов института ИТММ ННГУ в качестве дополнительных материалов к курсу «Параллельная обработка графов».

УДК 519.688
ББК 32.973

Содержание

Введение

Представление данных в виде графа используется во многих прикладных областях, однако обработка больших объемов таких данных на современных вычислительных системах выполняется менее эффективно, чем данных, полученных в ходе научного моделирования. Поэтому в научном и техническом сообществе большое внимание уделяется разработке эффективных реализаций алгоритмов на графах для высокопроизводительных систем. Это подтверждается большим числом научных публикаций, разработкой новых библиотек.

Разделение графов – одна из важных графовых задач, которая находит применение в различных прикладных областях. Поскольку в современных научных и технических приложениях, как правило, используются большие объемы данных, которые обрабатываются на суперкомпьютере, возникает проблема распределения данных по узлам вычислительной системы. Одно из важнейших применений разделения графов – распределить данные, полученные в ходе моделирования, между процессами таким образом, чтобы сократить межпроцессорные обмены. Поскольку этап распределения данных является подготовительным для выполнения других расчетов, важно выбрать для его решения алгоритм, позволяющий получить достаточно качественное решение за приемлемое время. Также задача разделения графов возникает при планировании транспортных и электросетей, разработке интегральных схем, в обработке изображений, моделировании биологических объектов. Общим для этих задач является необходимость разделить данные на группы, мало связанные между собой.

Разработка параллельных алгоритмов для разделения графов ведется с конца 1990х гг. и продолжается в настоящее время. Задача не теряет своей актуальности, поскольку объемы данных, к которым применяется разделение, все увеличиваются. Например, при математическом моделировании больших систем исходные данные могут не умещаться в память одного вычислительного узла, поэтому необходимо применять алгоритмы разделения, предназначенные для распределенных вычислительных систем. Кроме того, появление новых вычислительных архитектур ставит задачу модификации существующих алгоритмов для их эффективного использования.

В данном пособии приводятся практические примеры применения задачи разделения графа, дается математическая постановка задачи. Затем приводится краткий обзор методов ее решения и обзор специализированных библиотек. Далее подробно рассматривается один из методов разделения – многоуровневый метод, обсуждается его распараллеливание для систем с распределенной памятью. В последнем разделе дается пример использования функции разделения графов из библиотеки ParMETIS.

1. Задача разделения графов

1.1. Применение задачи разделения графов

Задача разделения графов возникает в различных прикладных областях. Приведем некоторые примеры [3].

1. *Параллельные вычисления в моделировании*

Одно из важнейших применений разделения графов – распределение данных между процессами для дальнейшей параллельной обработки. Например, при решении СЛАУ на кластерной системе матрица предварительно распределяется между процессами так, чтобы минимизировать межпроцессорные обмены. Если в ходе моделирования портрет матрицы СЛАУ не изменяется, то такая процедура выполняется один раз.

Алгоритмы разделения графов также используются для разделения сеток (mesh partitioning), представляющих собой декомпозицию некоторой расчетной области. Для моделирования процессов во многих естественно-научных областях выполняется дискретизация области моделирования, на которой решается система дифференциальных уравнений в частных производных (ДУвЧП). Если для численного решения ДУвЧП используется метод конечных разностей или метод конечных элементов, то строится СЛАУ с разреженной матрицей. При распределении сетки по вычислительным узлам строится граф, в котором одной вершине соответствует один или несколько узлов сетки.

Предварительное распределение графа по узлам вычислительной системы – один из этапов решения некоторых других задач на графах. Например, поиска в ширину, PageRank, поиска компонент связности, задач на собственные значения.

2. *Сложные сети*

Техники разделения графов применяются при планировании транспортных сетей. В этом случае строится граф, в котором вершинам соответствуют пересечения дорог, а ребрам – сегменты дорог (либо вершинам соответствуют города, а ребрам – пути, их соединяющие). Разделение графа можно использовать как один из подготовительных шагов для алгоритмов поиска кратчайших путей. Как правило, для этого применяется геометрическое разделение или многоуровневое разделение. Алгоритмы разделения также эффективно использовать, если необходим пересчет расстояний для графа дорог с изменяющимися весами ребер (customizable route planning).

Алгоритмы разделения графа находят применение при планировании электросетей. Так, возмущения и каскадные сбои являются одними из центральных проблем энергосистем, которые могут вызвать катастрофические отключения. Разделение единой электросети на

самодостаточные части может предотвратить распространение каскадного отказа.

В биологии для ряда задач применяются графовые модели. Например, это задачи взаимодействия протеинов, экспрессии генов (сети коэкспрессии генов). При построении графа для таких задач вершинам соответствуют биологические объекты, а ребрам – взаимодействие между ними в одном биологическом процессе. Разделение графа применяется для уменьшения размерности задачи и для выявления взаимодействия между кластерами объектов.

3. *Обработка изображений*

Алгоритмы разделения графов применяются в задаче сегментации изображения. Цель этой задачи – выделить на изображении объекты, то есть группы пикселей. Для этого по изображению строится граф, в котором вершинам соответствуют пиксели, а взвешенным ребрам – оценка их схожести (например, геодезическое расстояние между пикселями). Построенный таким образом граф имеет структуру, похожую на сетки (каждая вершина связана с 4-6 соседями).

4. *Разработка интегральных схем (VLSI desing)*

При разработке интегральной схемы алгоритмы разделения графов применяются для уменьшения сложности задачи. При построении графа по интегральной схеме вершинам соответствуют детали (блоки) схемы, а ребрам – соединяющие их провода. Схема разделяется на меньшие части, при этом длина соединяющих их проводов должна остаться приемлемой и короткой. Для задачи разделения интегральной схемы чаще применяются гиперграфы, поскольку провод схемы может соединять более двух деталей.

1.2. Постановка задачи

Пусть дан ненаправленный граф $G = (V, E)$ с неотрицательными весами ребер $w: E \to \boldsymbol{R}_{>0}$ и число $k \in \boldsymbol{N}_{>1}$. Тогда задача разделения графа заключается в нахождении разделения множества вершин V $\Pi = (V_1, V_2, \dots, V_k)$ на такие непересекающиеся подмножества, что:

$$V_i \cap V_j = \emptyset, \forall i \neq j$$

$$\bigcup_{i=1}^{k} V_k = V$$

Разделение должно быть *сбалансированным*, то есть все части V_i должны иметь близкий вес:

$$|V_i| \leq (1 + \varepsilon) \left\lceil \frac{|V|}{k} \right\rceil, \forall i \in \{1,2, \dots, k\}, \varepsilon \in \boldsymbol{R}_{\geq 0}$$

Вершина, смежная хотя бы с одной вершиной из другой части, называется *граничной*, а ребро, их соединяющее, *ребром разреза*.

$$u \in V_i, v \in V_j, (u, v) \in E, i \neq j$$

Обозначим множество граничных вершин B, множество ребер разреза – E_B.

Цель задачи разделения графа – найти разделение, для которого минимальна некоторая функция, оценивающая качество разделения. Как правило, качество разделения оценивается по весу ребер разреза: чем меньше суммарный вес ребер разреза, тем разделение лучше:

$$f: \sum_{(u,v)\in E_B} w(u,v) \to \min$$

Другая формулировка задачи разделения графа – минимизировать *максимальный объем коммуникаций*. Эта оценка используется, если разделение графа используется для распределения вершин графа по процессам. В задаче сегментации изображений, как правило, в качестве оценки качества разделения используется функция *нормализованного разреза*.

Задача нахождения сбалансированного разделения графа на k частей, минимизирующего функцию f, NP-полная. K-разделение можно получить из 2-разделения методом *рекурсивной бисекции*: граф делится на 2 части, затем, рекурсивно, первая часть делится на $\lfloor k/2 \rfloor$ частей, вторая – на $\lceil k/2 \rceil$ частей и т.д. Всего потребуется выполнить k – 1 бисекцию.

1.3. Методы разделения графов

Различные алгоритмы разделения графов разрабатываются с 1970-х годов. Большое внимание данной задаче уделялось в 1990х гг., связанное с развитием кластерных систем. Тогда были разработаны алгоритмы, наиболее используемые в настоящее время. Подробно с методами разделения графов и вспомогательными процедурами можно ознакомиться в сборнике [2]. Также обзор методов можно найти в работах [3, 9]. Перечислим основные из них.

Один из простых подходок к разделению графа основан на применении поиска в ширину. Это алгоритмы растущего разделения (graph growing partitioning), и жадного растущего разделения (greedy graph growing partitioning) которые применялись Джорджем и Лю (1981), Карписом и Кумаром (1998). Эти алгоритмы сейчас чаще применяются в качестве составной части более сложных схем. Другие алгоритмы используют геометрическую информацию о вершинах графа, если она доступна. Это алгоритм геометрического разделения (geometric partitioning), предложенный Симоном (1991), использованный Миллером, Тенгом и Вавасисом (1991) и др. Алгоритм спектрального разделения (spectral partitioning) использует для разделения информацию о собственных векторах графа. Этот алгоритм был предложен Донатом и Хоффманом (1973), использован Барнардом и Симоном (1993) и др.

В настоящее время широко применяется многоуровневое разделение графа (multilevel graph partitioning). Этот метод основан на получении разделения для маленького графа, сходного по структуре с исходным

графом, с целью проекции полученного решения на исходный граф. Алгоритм был предложен Хендриксоном и Леландом (1993), развит Карипсом и Кумаром (1996) и др. Алгоритм многоуровневого разделения графа будет подробно рассмотрен в следующем разделе.

Как и другие алгоритмы на графах, распараллеливание алгоритмов разделения имеет некоторые особенности. Так, при увеличении числа процессов (потоков) качество разделения, как правило, снижается. При построении параллельного алгоритма требуется большой объем синхронизаций между потоками (процессами) либо увеличение числа перезапусков алгоритма. На практике используются модификации последовательных алгоритмов, дающие достаточное качество разделения за приемлемое время.

1.4. Библиотеки для разделения графов

Поскольку задача разделения графа возникает во многих предметных областях, в настоящее время разработано большое число специализированных библиотек, решающих эту задачу. Большинство из них имеют последовательную версию и версию для систем с распределенной памятью. Перечислим некоторые из них.

Chaco (http://swmath.org/software/9640) – одна из первых библиотек для разделения графов, в ней реализованы многоуровневый и спектральный методы разделения. В настоящее время не поддерживается.

Библиотеки Metis (http://glaros.dtc.umn.edu/gkhome/views/metis) и Scotch (https://www.labri.fr/perso/pelegrin/scotch/) – одни из самых используемых библиотек, в которых реализован многоуровневый метод. Эти библиотеки ориентированы на графы и сетки, полученные в ходе математического моделирования. У них есть версии для систем с распределенной памятью (ParMETIS, PT-Scotch), METIS также имеет параллельную реализацию для систем с общей памятью (mt-metis).

KaHIP (http://algo2.iti.kit.edu/kahip/) – библиотека, в которой реализован многоуровневый метод и методы разделения на основе алгоритмов для потоков в сетях. Отличительной особенностью библиотеки являются эвристические алгоритмы, ориентированные на разделение неструктурированных графов – графов социальных сетей, веб-графов. Имеет реализацию для систем с распределенной памятью.

Jostle (http://chriswalshaw.co.uk/jostle/) – пакет для разделения неструктурированных сеток, имеет последовательную версию и версию для систем с распределенной памятью. В нем также реализован многоуровневый метод разделения и алгоритм балансировки на основе диффузии. В настоящее время коммерциализован под названием NetWorks.

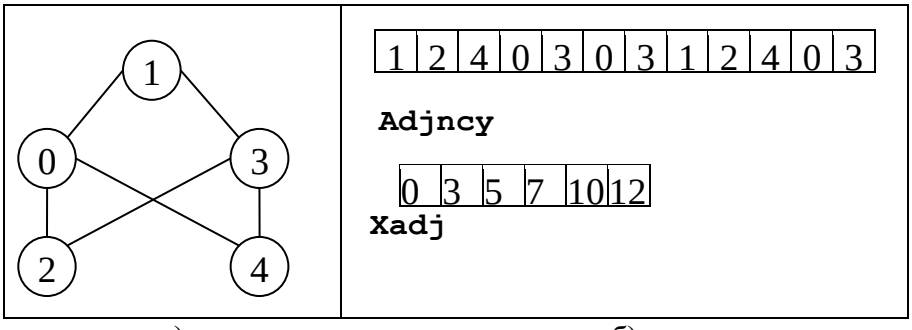
2. Многоуровневый алгоритм разделения графов

2.1. Хранение графов

При реализации алгоритмов на графах, как правило, используется представление графов в виде матрицы смежности или списков смежности. Если в графе число ребер пропорционально числу вершин, то граф называется *разреженным* и его целесообразно хранить в виде *списков смежности*. Для этого используется два массива. Пусть граф $G = (V, E)$ содержит N вершин и M ребер. Первый массив, **`Adjncy`**, хранит номера вершин, связанных с данной вершиной, последовательно для вершины 0, вершины 1, …, вершины $N-1$. Второй массив, **`Xadj`**, хранит индексы начала списка смежности каждой вершины в массиве **`Adjncy`**. Дополнительно в массиве **`Xadj`** хранится «фиктивный» элемент – индекс последнего элемента массива **`Adjncy`**, увеличенный на единицу. Таким образом, все вершины, смежные с вершиной i, хранятся в массиве **`Adjncy`** по индексам от **`Xadj[i]`** до **`Xadj[i+1]-1`** включительно. Для графа с N вершинами и M ребрами массив **`Adjncy`** имеет размер 2*M (поскольку каждое ребро графа хранится в этом массиве два раза), массив **`Xadj`** – размер $N+1$. На рис. 1 показан пример представления графа в виде списков смежности.

Если в графе заданы веса ребер, то для разреженного графа необходимо хранить дополнительный массив размера $2M$, в котором записаны веса ребер соответственно их порядку в массиве **`Adjncy`**. Если в графе заданы веса вершин, то они хранятся в дополнительном массиве размера N.

В вычислениях на распределенной памяти, когда хранение графа разделено между несколькими процессами, используются *распределенные списки смежности*. При этом, как правило, предполагается, что процессы хранят информацию о подряд идущих вершинах. Например, для графа с рисунка 1, нулевой процесс хранит списки смежности для вершин 0, 1, 2, первый – для вершин 3, 4. На каждом процессе хранятся массивы списков смежности **`Adjncy`** и **`Xadj`** для его локальных вершин, нумерация в массиве **`Xadj`** на каждом процессе начинается с 0.



а) б)

Рис. 1. Хранение графа в формате CRS.
а) Исходный граф, б) соответствующие графу списки смежности

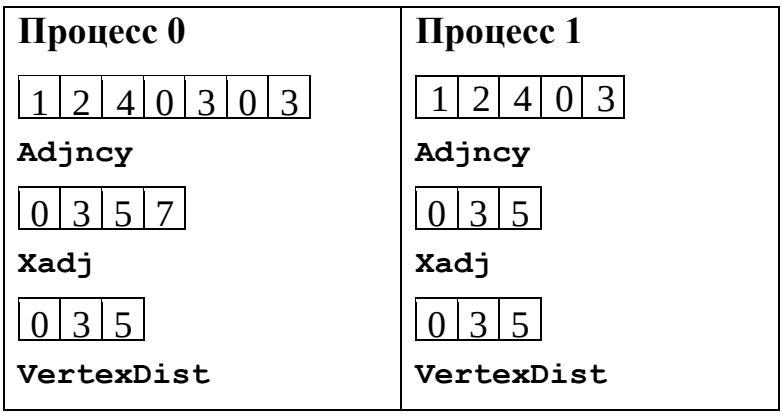


Рис. 2. Хранение графа, изображенного на рис. 1, на двух процессах в виде распределенных списков смежности. Процесс 0 хранит списки смежности вершин 0, 1, 2, процесс 1 – списки смежности вершин 3 и 4

Дополнительно каждый процесс хранит массив **`VertexDist`**, который описывает распределение вершин между процессами. Аналогично массиву **`Xadj`**, массив **`VertexDist`** имеет размер p+1, где p – число процессов, и для процесса с номером i локальными являются вершины с **`VertexDist[i]`** по **`VertexDist[i+1]-1`** включительно. На рис. 2 показан пример хранения графа, изображенного на рис. 1, на двух процессах.

2.2. Последовательный алгоритм разделения графа

При использовании многоуровневого метода разделение вычисляется в три этапа: огрубление, разделение, развертывание. Рассмотрим разделение графа $G = G_1(V_1, E_1)$.

1. *Огрубление графа* (*coarsening*)

На этом этапе выполняется построение последовательности сжимающихся графов $G_1(V_1, E_1)$, $G_2(V_2, E_2)$, …, $G_m(V_m, E_m)$. При этом число вершин каждого следующего графа меньше, чем у предыдущего: $|V_1| > |V_2| > \ldots > |V_m|$, и структура каждого следующего в последовательности графа отражает структуру предыдущего. Для огрубления графов, полученных в ходе научного моделирования, используются алгоритмы поиска максимального паросочетания в графе, тот есть такого наибольшего множества пар вершин, в котором ребра, их соединяющие, не инцидентны друг другу. Это алгоритмы случайных паросочетаний (random matching), паросочетания тяжелых ребер (heavy edge matching), паросочетания тяжелых клик (heavy clique matching, HCM) и др. Применение алгоритма максимального паросочетания позволяет построить граф с числом вершин до 2 раз меньше исходного. Описание алгоритмов паросочетания можно найти, например, в работе [5]. Для графов с нерегулярной структурой, графов социальных сетей используются алгоритмы кластеризации, например, метод распространения меток [10]. Такие алгоритмы позволяют объединить более двух вершин в одну супервершину.

2. *Разделение графа (partitioning)*

На этом этапе выполняется разделение наименьшего графа G_m на k частей. На этом этапе, как правило, используется вспомогательный алгоритм разделения.

3. *Развертывание графа (uncoarsening)*

На этом этапе выполняется проекция разделения графа G_m на исходный граф G через последовательность графов G_m, G_{m-1}, ..., G_1. На каждом шаге i выполняется проекция разделения графа G_{m-i+1} на граф G_{m-i}, затем выполняется улучшение разделения графа G_{m-i} с помощью алгоритма локальной оптимизации. Цель работы таких алгоритмов – уменьшить разрез и улучшить баланс полученных частей. Как правило, для этого используются модификации метода Кернигана–Лина, генетические алгоритмы, алгоритмы минимизации разреза для потоков в сетях, метод диффузии.

Приведем принцип работы алгоритма улучшения разделения на основе метода Кернигана–Лина, используемый в ParMETIS [5]. Центральной идеей алгоритма является использование функции, которая для каждой вершины оценивает уменьшение разреза графа при перемещении этой вершины в другую часть разделения. Пусть вершина v находилась в части разделения P_v, тогда ее можно переместить в одну из частей, в которых находятся смежные с ней вершины. Обозначим это множество PN. Тогда если вершина была внутренняя для разделения, $PN = P_v$, ее перемещение не повлияет на разрез и баланс разделения. Если вершина v была граничная, $PN \neq P_v$, хотя бы одна из соседних с ней вершин находится в другой части разделения, то функция $gain$ перемещения v в часть P определяется как разница суммы весов «внешних» и «внутренних» ребер для части разделения P_v:

$$gain(v, P) = \sum_{z \in P} w(v, z) - \sum_{u \in P_v} w(v, u)$$

Алгоритм улучшения разделения выполняется итерационно. На каждой итерации все вершины посещаются в случайном порядке и проверяются на целесообразность перемещения, согласно значению функции $gain$. Если вершина граничная, то она перемещается в ту часть, для которой уменьшение разреза будет наибольшим. Если перемещение v не позволяет уменьшить разрез, то эту вершину можно переместить в другую часть с целью улучшения баланса разделения. После выполнения перемещения пересчитывается значение $gain$ для смежных с v вершин.

Для улучшения качества разделения, полученного многоуровневым методом, также применяется итерационное уточнение. Для этого многоуровневая процедура выполняется несколько раз, и на каждой следующей итерации учитываются результаты предыдущего разделения. Одна из возможных реализаций такого подхода – при огрублении не объединять вершины, если соединяющее их ребро было ребром разреза на предыдущей итерации.

2.3. Параллельный алгоритм разделения графа

Многоуровневый алгоритм разделения графа широко используется в параллельных вычислениях, на его основе выполнены реализации в большинстве специализированных библиотек. Рассмотрим этот алгоритм для графа с n вершинами, исполняемый на p процессах, для $k = p$. Предполагается, что граф разделен между процессами «полосами», и на каждом процессе хранится n/p вершин с их списками смежности.

Ниже приведен общий подход к распараллеливанию на основе реализации из библиотеки ParMETIS [5]. В ряде библиотек, в том числе, в ParMETIS, используется вспомогательный этап – вычисление раскраски графа. Его результаты используются на этапах огрубления и улучшения разделения для сокращения межпроцессорного обмена.

1. *Огрубление*

На этом этапе все p процессов выполняют поиск максимального паросочетания. Предположим, что для графа уже найдена раскраска в c цветов. Огрубление выполняется итерационно, при этом на итерации i каждый процесс ищет пары для своих локальных вершин цвета i, которые еще не вошли в паросочетание. После того, как для каждой вершины выбирается пара-кандидат, выполняется синхронизация. Процессы обмениваются списками-«запросами». Запрос на объединение принимается, если вершина с чужого процесса была свободна или она отправила такой же запрос на объединение. Иначе запрос отклоняется. Результаты обработки запросов рассылаются процессам-отправителям, которые корректируют свое паросочетание.

После построения паросочетания определяется, как огрубленный граф будет распределен между процессами. Каждый процесс получает и отправляет списки смежности вершин, которые необходимы для построения локальной части огрубленного графа. Процесс огрубления заканчивается, когда будет получен граф с $O(p)$ вершинами.

2. *Разделение*

Поскольку на этапе разделения граф G_m содержит небольшое число вершин, начальное разделение можно выполнить независимо на каждом процессе, а затем выбрать лучшее разделение. Другой подход – выполнить параллельное разделение графа на p частей методом рекурсивной бисекции. При этом каждый процесс будет выполнять вычисления по одному пути в дереве рекурсии. В результате этого шага каждый процесс будет хранить все вершины одной из p частей разделения.

3. *Улучшение разделения*

На этом этапе все p процессов выполняют итерационное улучшение разделения. Для того, чтобы выделить группы вершин, которые можно перемещать в другие части разделения одновременно, используется раскраска графа. Алгоритм выполняется за c итераций, где c – число цветов в раскраске графа. На итерации i алгоритма выполняются перемещения для вершин цвета i. Поскольку вершины одного цвета образуют независимое

множество, все их можно переместить одновременно. Затем необходимо пересчитать $gain$ для всех вершин, смежных с перемещенными. В конце работы алгоритма выполняется коммуникация между процессами, в результате которой все вершины из части разделения i перемещаются на процесс i.

3. Пример использования библиотеки ParMETIS

3.1. Сборка библиотеки

Библиотека ParMETIS предоставляется в исходных кодах, может быть использована под ОС Linux и Windows. Дистрибутив и краткую инструкцию по установке библиотеки можно найти на странице [7], а также в файле Install.txt из архива с исходным кодом. Для установки необходимо иметь компилятор языка C с поддержкой C99, а также CMake v. 2.8 или старше.

Для того, чтобы собрать библиотеку для ОС Linux, необходимо выполнить следующее:

1. Загрузить архив с библиотекой со страницы [7]. Например, parmetis-4.0.3.tar.gz

2. Распаковать в нужной папке, выполнив команды:

```
gunzip parmetis-4.0.3.tar.gz
tar -xvf parmetis-4.0.3.tar
```

В результате появится папка parmetis-4.0.3.

3. Установить размер типов данных, используемых в реализации. Для этого в файле parmetis-4.0.3/metis/include/metis.h необходимо указать значение констант IDXTYPEWIDTH для целочисленого типа и REALTYPEWIDTH для типа с плавающей точкой. Библиотека поддерживает размер 32 для 32х-битных архитектур и размеры 32 и 64 для 64х-битных архитектур. В реализации используются знаковые 32х или 64х-битные целочисленные типы, вещественный тип одинарной или двойной точности. Например,

```
#define IDXTYPEWIDTH 32
#define REALTYPEWIDTH 64
```

4. Перейти в папку parmetis-4.0.3. Сконфигурировать библиотеку командой

```
make config
```

На этом этапе можно указать дополнительные опции сборки, – компилятор (опции cc, cxx), debug или release версия (debug = 1), необходимость проверок (assert = 1). Полный список опций сборки указан в файле Build.txt папки parmetis-4.0.3. Например,

```
make config cc=icc
```

5. Скомпилировать библиотеку командой

```
make
```

В результате в папке parmetis-4.0.3 появится папка build/<название сборки>, которая будет включать следующие папки:

- libmetis, содержащая файл библиотеки metis.a
- libparmetis, содержащая файл библиотеки parmetis.a

- programs – папка, содержащая примеры исполняемых файлов. Исходный код этих примеров находится в папке programs.

3.2. Прототип функции разделения

Для использования библиотечных функций граф необходимо представить в распределенном CRS формате, который был описан в разделе 2.1. Для графа можно задать веса ребер и веса вершин.

Чтобы использовать многоуровневый алгоритм k-разделения графа, необходимо вызвать функцию **ParMETIS_V3_PartKway** со следующим прототипом [4]:

```
int ParMETIS_V3_PartKway (idx_t *vtxdist, idx_t *xadj,
idx_t *adjncy, idx_t *vwgt, idx_t *adjwgt, idx_t *wgtflag,
idx_t *numflag, idx_t *ncon, idx_t *nparts, real t *tpwgts,
real_t *ubvec, idx_t *options, idx_t *edgecut, idx_t *part,
MPI Comm *comm)
```

Здесь **idx_t** – целочисленный тип, **real_t** – вещественный тип.

Параметры функции обозначают следующее:

- **vtxdist** – массив, описывающий, как вершины распределены между процессами.
- **xadj**, **adjncy** – массивы, описывающие списки смежности для локальных для процесса вершин.
- **vwgt** – массив весов вершин. Для каждой вершины можно задать многомерные веса. В этом случае, в массив записываются сначала все веса первой размерности, затем второй и т.д.
- **adjwgt** – массив весов ребер.
- **wgtflag** – флаг, показывающий наличие весов у графа. Возможные значения: 0 – граф без весов, 1 – заданы только веса ребер, 2 – заданы только веса вершин, 3 – заданы веса и вершин, и ребер.
- **numflag** – флаг, показывающий тип нумерации в **массивах vtxdist**, **xadj**, **adjncy**, **part**. Возможные значения: 0 – нумерация с 0, 1 – нумерация с 1.
- **ncon** – размерность весов, ассоциированных с каждой вершиной, а также число ограничений для определения баланса разделения.
- **nparts** – число частей, на которые необходимо разделить граф.
- **tpwgts** – массив размером ncon × nparts, который описывает распределение весов вершин для удовлетворения условию баланса частей. Если в искомом разделении все части должны иметь близкий вес, то все элементы массива задаются как 1/nparts. Сумма элементов массива **tpwgts** должна равняться 1.
- **ubvec** – массив размера **ncon**, который содержит допустимый дисбаланс частей разделения для каждой размерности весов вершин. Рекомендуемое значение элементов массива – 1.05.

- **`options`** – массив опций. Элемент **`options[0]`** обозначает, будут ли использоваться параметры по умолчанию (значение 0) или заданные пользователем (значение 1). В последнем случае будут учитываться значения следующих элементов массива. Элемент **`options[1]`** задает количество дополнительной информации, возвращаемой алгоритмом. Значение этого параметра задается как сумма констант, указанных в файле parmetis.h. При **`options[1] = 1`** будет выведена информация о времени работы алгоритма. Элемент **`options[2]`** задает значение ядра генератора случайных чисел.
- **`edgecut`** – разрез полученного разделения.
- **`part`** – результирующий массив с разделением. По завершении вычислений на каждом процессе элемент массива **`part[i]`** будет содержать номер части разделения для локальной вершины i.
- **`comm`** – MPI коммуникатор.

Функция возвращает код ошибки. Возможные значения: **`METIS_OK`** – вычисление успешно, **`METIS_ERROR`** – ошибка в ходе работы алгоритма.

3.3. Пример

Приведем пример использования разделения графа.

В рамках одного из научных проектов в ННГУ был разработан программный комплекс «CardioModel» [8], предназначенный для моделирования электрофизиологической активности сердца человека на суперкомпьютере. Целью вычислений является моделирование нормальной или атипичной электрофизиологической деятельности сердца (например, инфаркта) для конкретного пациента. В качестве исходных данных используются результаты компьютерных и магнитно-резонансных томограмм пациента, по которым строится трехмерная модель сердца. В результате дискретизации сердце представляется в виде тетраэдральной сетки, в которой узлы – клетки сердечной ткани разного типа. Сетка хранится распределенно, поскольку модель высокой точности имеет порядка 10^7 узлов. В ходе моделирования выполняется многократное решение дифференциальных уравнений в частных производных методом конечных элементов. Численно это заключается в решении СЛАУ с разреженной матрицей. В пакете «CardioModel» для этого используется итерационный решатель из библиотеки PETSc (http://www.mcs.anl.gov/petsc). Применение алгоритма разделения графа на подготовительном этапе вычислений позволяет уменьшить время решения СЛАУ за счет сокращения коммуникаций между процессами.

Таким образом, общая вычислительная схема состоит из следующих этапов:

1. Чтение исходных данных (тетраэдральной сетки).
2. Построение матрицы для решения СЛАУ методом конечных элементов.

3. Разделение матрицы с помощью ParMETIS.
4. Перераспределение матрицы по вычислительным узлам.
5. Численное моделирование с использованием решателя PETSc.

```
#include "parmetis.h"
#include "petsc.h"

int main(int argc, char** argv)
{
  int* locPartition;
  int* globPartition;
  Mesh mesh;
  CRSmatrix A;
  CRSmatrix A2;
  double* v;
  int* weights;

  // 1. Чтение сетки
  Read(&mesh);
  // 2. Построение матрицы
  SetUpMatrix(mesh, &A);
  // 3. Разделение матрицы
  Partitioning(A.vtxdist, A.row, A.column, weights,
     locPartition, globalPartition);
  // 4. Перераспределение матрицы
  ApplyPartition(A, &A2, globalPartition);
  // 5. Численное моделирование
  Solve(A2, v);
}
```

Рассмотрим функцию разделения:

```
int Partitioning(int* vtxdist, int* xadj, int* adjncy,
                 int* vertexWeights, int* locPartition,
                 int* globalPartition)
```

Функция **Partitioning** принимает на вход граф, распределенный между процессами, согласно массиву **vtxdist**. Массивы **xadj** и **adjncy** хранят списки смежности, локальные для процесса, массив **vertexWeights** хранит веса локальных вершин. В рассматриваемом примере графом является CRS-представление разреженной матрицы. Выходные параметры функции – массив **locPartition**, хранящий локальную для процесса часть разделения, и массив **globalPartition**, в который со всех процессов собирается итоговое разделение. Функция возвращает код ошибки, полученный в ходе работы функции **ParMETIS_V3_PartKway**. В данном примере используются параметры разделения по умолчанию.

```
#include "parmetis.h"

int Partitioning(int* vtxdist, int* xadj, int* adjncy,
                 int* vertexWeights, int* locPartition,
                 int* globalPartition)
{
    MPI_Comm comm = MPI_COMM_WORLD;
    int nProcs;  // число процессов
    int procID;  // номер процесса в коммуникаторе
    int* nRecv;  // массив для сбора разделения со всех
процессов
    // параметры библиотеки
    int numflag = 0;
    int edgecut;
    int wgtflag = 2; // заданы только веса вершин
    int ncon = 1;
    real_t* tpwgts;
    real_t ubvec = 1.05;
    int options[10];
    int errorCode;

    MPI_Comm_size(comm, &nProcs);
    MPI_Comm_rank(comm, &procID);

    // установка параметров по умолчанию
    tpwgts = new real_t [ncon * nProcs];
    for (int i = 0; i < nProcs; i++)
        tpwgts[i] = 1.0 / (real_t)nProcs;
    options[0] = 0;

    // разделение
    errorCode = ParMETIS_V3_PartKway(vtxdist, xadj, adjncy,
      vertexWeights, NULL,  &wgtflag, &numflag, &ncon, &nProcs,
      tpwgts, &ubvec, options, &edgecut, locPartition, &comm);

    // сбор глобального разделения
    if (errorCode == METIS_OK)
    {
      nRecv = new int [nProcs];
      for(int i = 0; i < nProcs; i++)
        nRecv[i] = vtxdist[i + 1] - vtxdist[i];
      MPI_Allgatherv(locPartition, nRecv[procID], MPI_INT,
        globalPartition, nRecv, vtxdist, MPI_INT, comm);

      delete [] nRecv;
    }

    delete [] tpwgts;

    return errorCode;
}
```

Литература

1. Abou-Rjeili A., Karypis G. Multilevel Algorithms for Partitioning Power-Law Graphs // Procs. of 20th IPDPS. – IEEE, 2006. – P. 103.
2. Bichot C. E., Siarry P. (ed.). Graph partitioning. – ISTE, 2011.
3. Buluç A. et al. Recent Advances in Graph Partitioning // Algorithm Engineering. – Springer, Cham, 2016. – P. 117–158.
4. Karipis G., Schloegel K. ParMETIS Manual. Parallel Graph Partitioning and Sparse Matrix Ordering Library. Version 4.0. // Technical report, U. of Minnesota, Dep. of Computer Science and Engineering. – 2011. URL: http://glaros.dtc.umn.edu/gkhome/fetch/sw/parmetis/manual.pdf
5. Karypis G., Kumar V. Parallel Multilevel Series k-way Partitioning Scheme for Irregular Graphs // Siam Review. – 1999. – Vol. 41. – №. 2. – P. 278–300.
6. Meyerhenke H., Sanders P., Schulz C. Parallel Graph Partitioning for Complex Networks // IEEE Transactions on Parallel and Distributed Systems. – 2017. – Vol. 28. – №. 9. – P. 2625–2638.
7. ParMETIS Download page. – URL: http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download
8. Petrov V. et al. CardioModel–New Software for Cardiac Electrophysiology Simulation // Russian Supercomputing Days. LNCS. – Springer, Cham, 2018. – P. 195–207.
9. Pothen A. Graph partitioning algorithms with applications to scientific computing // Parallel Numerical Algorithms. – Springer, Dordrecht, 1997. – P. 323–368.
10. Ugander J., Backstrom L. Balanced Label Propagation for Partitioning Massive Graphs // 6’th Int. Conf. on Web Search and Data Mining (WSDM’13). – ACM, 2013. – P. 507–516.

Анна Юрьевна **Пирова**

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ РАЗДЕЛЕНИЯ ГРАФОВ

Учебно-методическое пособие

Компьютерная верстка – А.Ю. Пирова

Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского».
603950, Нижний Новгород, пр. Гагарина, 23.