Высокопроизводительные параллельные вычисления

Лекция №8
Тема: Параллельные методы на графах

Козбакова Айнур Холдасовна– PhD, профессор, кафедра «Информационные системы» Факультет «Инжиниринг и информационных технологии» 01.09.2025 г.

Рассмотрим способы параллельной реализации алгоритмов на *графах* на примере *задачи поиска кратчайших путей* между всеми парами пунктов назначения и *задачи выделения минимального охватывающего дерева* (*остова*) *графа*. Кроме того, рассмотрим *задачу оптимального разделения графов*, широко используемую для организации параллельных вычислений. Для представления *графов* при рассмотрении всех перечисленных задач будут применяться матрицы смежности.

Задача поиска всех кратчайших путей

Исходной информацией для задачи является взвешенный *граф* G=(V,R), содержащий n вершин (|V|=n), в котором каждому ребру *графа* приписан неотрицательный вес. *Граф* будем полагать *ориентированным*, т.е. если из вершины i есть ребро в вершину j, то из этого не следует наличие ребра из j в i. В случае если вершины все же соединены взаимообратными ребрами, веса, приписанные им, могут не совпадать. Рассмотрим задачу, в которой для имеющегося *графа* G требуется найти минимальные длины путей между каждой парой вершин *графа*. В качестве практического примера можно привести задачу составления маршрута движения транспорта между различными городами при заданном расстоянии между населенными пунктами и другие подобные задачи.

В качестве метода, решающего задачу поиска кратчайших путей между всеми парами пунктов назначения, далее используется *алгоритм Флойда.*.

Последовательный алгоритм Флойда.

Для поиска минимальных расстояний между всеми парами пунктов назначения Флойд предложил алгоритм, сложность которого имеет порядок n^3. В общем виде данный алгоритм может быть представлен следующим образом:

```
// Последовательный алгоритм Флойда
for (k = 0; k < n; k++)
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      A[i, j] = min(A[i, j], A[i, k] + A[k, j]);
```

(реализация операции выбора минимального значения min должна учитывать способ указания в матрице смежности несуществующих дуг *графа*). Как можно заметить, в ходе выполнения алгоритма матрица смежности A изменяется, после завершения вычислений в матрице A будет храниться требуемый результат – длины минимальных путей для каждой пары вершин исходного *графа*.

Разделение вычислений на независимые части

Как следует из общей схемы *алгоритма Флойда*, основная вычислительная нагрузка при решении задачи поиска кратчайших путей состоит в выполнении операции выбора минимальных значений. Данная операция является достаточно простой, и ее распараллеливание не приведет к заметному ускорению вычислений. Более эффективный способ организации параллельных вычислений может состоять в одновременном выполнении нескольких операций обновления значений матрицы A.

Покажем корректность такого способа организации параллелизма. Для этого нужно доказать, что операции обновления значений матрицы A на одной и той же итерации внешнего цикла k могут выполняться независимо. Иными словами, следует показать, что на итерации k не происходит изменения элементов A_{ik} и A_{kj} ни для одной пары индексов (i, j). Рассмотрим выражение, по которому происходит изменение элементов матрицы A:

$A_{ij} \leftarrow \min (A_{ij}, A_{ik} + A_{kj})$.

Для i=k получим

$A_{kj} \leftarrow \min (A_{kj}, A_{kk} + A_{kj})$,

но тогда значение A_{kj} не изменится, т.к. $A_{kk}=0$.

Для j=k выражение преобразуется к виду

$A_{ik} \leftarrow \min (A_{ik}, A_{ik} + A_{kk})$,

что также показывает неизменность значений A_{ik}. Как результат, необходимые условия для организации параллельных вычислений обеспечены, и, тем самым, в качестве *базовой подзадачи* может быть использована операция обновления элементов матрицы A (для указания подзадач будем применять индексы обновляемых в подзадачах элементов).

Выделение информационных зависимостей

Выполнение вычислений в подзадачах становится возможным только тогда, когда каждая подзадача (i, j) содержит необходимые для расчетов элементы A_{ij}, A_{ik}, A_{kj} матрицы A. Для исключения дублирования данных разместим в подзадаче (i, j) единственный элемент A_{ij}, тогда получение всех остальных необходимых значений может быть обеспечено только при помощи передачи данных. Таким образом, каждый элемент A_{kj} строки k матрицы A должен быть передан всем подзадачам (k, j), $1 \le j \le n$, а каждый элемент A_{ik} столбца k матрицы A должен

быть передан всем подзадачам (i, k), $1 \le i \le n$.

Масштабирование и распределение подзадач по процессорам

Как правило, число доступных процессоров p существенно меньше, чем число базовых задач n^2 ($p << n^2$). Возможный способ укрупнения вычислений состоит в использовании *ленточной схемы* разбиения матрицы A – такой подход соответствует объединению в рамках одной базовой подзадачи вычислений, связанных с обновлением элементов одной или нескольких строк (*горизонтальное* разбиение) или столбцов (*вертикальное* разбиение) матрицы A. Эти два типа разбиения практически равноправны – учитывая дополнительный момент, что для алгоритмического языка C массивы располагаются по строкам, будем рассматривать далее только разбиение матрицы A на горизонтальные полосы. Следует отметить, что при таком способе разбиения данных на каждой итерации *алгоритма Флойда* потребуется передавать между подзадачами только элементы одной из строк матрицы A. Для оптимального выполнения подобной коммуникационной операции топология сети должна обеспечивать эффективное представление структуры сети передачи данных в виде гиперкуба или полного *графа*.

Задача нахождения минимального охватывающего дерева

Охватывающим деревом (или *остовом*) неориентированного *графа* G называется подграф T *графа* G, который является деревом и содержит все вершины из G. Определив вес подграфа для взвешенного *графа* как сумму весов входящих в подграф дуг, под *минимально охватывающим деревом* (МОД) T будем понимать охватывающее дерево минимального веса. Содержательная интерпретация задачи нахождения МОД может состоять, например, в практическом примере построения локальной сети персональных компьютеров с прокладыванием соединительных линий связи минимальной длины.

Последовательный алгоритм Прима

Алгоритм начинает работу с произвольной вершины *графа*, выбираемой в качестве корня дерева, и в ходе последовательно выполняемых итераций расширяет конструируемое дерево до МОД. Пусть V_T есть множество вершин, уже включенных алгоритмом в МОД, а величины d_i, $1 \le i \le n$, характеризуют дуги минимальной длины от вершин, еще не включенных в дерево, до множества V_T, т.е.

$$\forall i \notin V_T \Rightarrow d_i = \min\{w(i,u) : u \in V_T, (i,u) \in R\}$$

(если для какой-либо вершины $i \notin V_T$ не существует ни одной дуги в V_T, значение d_i устанавливается равным ∞). В начале работы алгоритма выбирается корневая вершина МОД s и полагается $V_T=\{s\}$, $d_s=0$.

Действия, выполняемые на каждой итерации *алгоритма Прима*, состоят в следующем:

- определяются значения величин d_i для всех вершин, еще не включенных в состав МОД;
- выбирается вершина t *графа* G, имеющая дугу минимального веса до множества V_T $t:d_t$, $i \notin V_T$;
- вершина t включается в V_T.

После выполнения n-1 итерации метода МОД будет сформировано. Вес этого дерева может быть получен при помощи выражения

$$W_T = \sum_{i=1}^{n} d_i.$$

Трудоемкость нахождения МОД характеризуется квадратичной зависимостью от числа вершин *графа* $T_1 \sim n^2$.

Разделение вычислений на независимые части

Оценим возможности параллельного выполнения рассмотренного алгоритма нахождения минимально охватывающего дерева.

Итерации метода должны выполняться последовательно и, тем самым, не могут быть распараллелены. С другой стороны, выполняемые на каждой итерации алгоритма действия являются независимыми и могут реализовываться одновременно. Так, например, определение величин d_i может осуществляться для каждой вершины *графа* в отдельности, нахождение дуги минимального веса может быть реализовано по каскадной схеме и т.д.Распределение данных между процессорами вычислительной системы должно обеспечивать независимость перечисленных операций *алгоритма Прима*. В частности, это может быть реализовано, если каждая вершина *графа* располагается на процессоре вместе со всей связанной с вершиной информацией. Соблюдение данного принципа приводит к тому, что при равномерной загрузке каждый процессор p_j, $1 \leq j \leq p$, должен содержать:

- набор вершин
  $V_j = \{v_{i_j+1}, v_{i_j+2}, \ldots, v_{i_j+k}\},\ i_j = k \cdot (j-1), k = \lceil n/p \rceil;$
- соответствующий этому набору блок из k величин
  $\Delta_j = \{d_{i_j+1}, d_{i_j+2}, \ldots, d_{i_j+k}\};$
- вертикальную полосу матрицы смежности *графа* G из k соседних столбцов
  $A_j = \{\alpha_{i_j+1}, \alpha_{i_j+2}, \ldots, \alpha_{i_j+k}\} (\alpha_s \text{ есть } s\text{-й столбец матрицы } A);$
- общую часть набора V_j и формируемого в процессе вычислений множества вершин V_T.

Как итог можем заключить, что базовой подзадачей в параллельном *алгоритме Прима* может служить процедура вычисления блока значений Δ_j для вершин V_j матрицы смежности A *графа* G.

Выделение информационных зависимостей

С учетом выбора базовых подзадач общая схема параллельного выполнения *алгоритма Прима* будет состоять в следующем:

- определяется вершина t *графа* G, имеющая дугу минимального веса до множества V_T. Для выбора такой вершины необходимо осуществить поиск минимума в наборах величин d_i, имеющихся на каждом из процессоров, и выполнить сборку полученных значений на одном из процессоров;
- номер выбранной вершины для включения в охватывающее дерево передается всем процессорам;
- обновляются наборы величин d_i с учетом добавления новой вершины.

Таким образом, в ходе параллельных вычислений между процессорами выполняются два типа информационных взаимодействий: сбор данных от всех процессоров на одном из процессоров и передача сообщений от одного процессора всем процессорам вычислительной системы.

Масштабирование и распределение подзадач по процессорам

По определению количество базовых подзадач всегда соответствует числу имеющихся процессоров, и, тем самым, проблема масштабирования для параллельного алгоритма не возникает. Распределение подзадач между процессорами должно учитывать характер выполняемых в *алгоритме Прима* коммуникационных операций. Для оптимальной реализации требуемых информационных взаимодействий между базовыми подзадачами топология сети передачи данных должна обеспечивать эффективное представление в виде гиперкуба или полного *графа*.

Задача оптимального разделения графов

Проблема оптимального разделения графов относится к числу часто возникающих задач при проведении различных научных исследований, использующих параллельные вычисления. В качестве примера можно привести задачи обработки данных, в которых области расчетов аппроксимируются двумерными или трехмерными вычислительными сетками. Получение результатов в таких задачах сводится, как правило, к выполнению тех или иных процедур обработки для каждого элемента (узла) сети. При этом в ходе вычислений между соседними элементами сети может происходить передача результатов обработки и т.п. Эффективное решение таких задач на многопроцессорных системах с распределенной памятью предполагает разделение сети между процессорами таким образом, чтобы каждому из процессоров выделялось примерно равное число элементов сети, а межпроцессорные коммуникации, необходимые для выполнения информационного обмена между соседними элементами, были минимальными. Очевидно, что такие задачи разделения сети между процессорами могут быть сведены к *проблеме оптимального разделения графа*. Данный подход целесообразен, потому что представление модели вычислений в виде *графа* позволяет легче решить вопросы хранения обрабатываемых данных и предоставляет возможность применения типовых алгоритмов обработки *графов*.

Для представления сети в виде графа каждому элементу сети можно поставить в соответствие вершину графа, а дуги графа использовать для отражения свойства близости элементов сети (например, определять дуги между вершинами графа тогда и только тогда, когда соответствующие элементы исходной сети являются соседними).

Постановка задачи оптимального разделения графов

Пусть дан взвешенный неориентированный *граф* $G=(V,E)$, каждой вершине $v \in V$ и каждому ребру $e \in E$ которого приписан вес. *Задача оптимального разделения графа* состоит в разбиении его вершин на непересекающиеся подмножества с максимально близкими суммарными весами вершин и минимальным суммарным весом ребер, проходящих между полученными подмножествами вершин.

Следует отметить возможную противоречивость указанных критериев разбиения *графа* – равновесность подмножеств вершин может не соответствовать минимальности весов граничных ребер и наоборот. В большинстве случаев необходимым является выбор того или иного компромиссного решения. Так, в случае невысокой доли коммуникаций может оказаться эффективной оптимизация веса ребер только среди решений, обеспечивающих оптимальное разбиение множества вершин по весу. Для простоты изложения будем полагать веса вершин и ребер *графа* равными единице.

Метод рекурсивного деления пополам

Для решения задачи разбиения *графа* можно рекурсивно применить *метод бинарного деления*, при котором на первой итерации *граф* разделяется на две равные части, далее на втором шаге каждая из полученных частей также разбивается на две части и т. д. В данном подходе для разбиения *графа* на k частей необходимо $\log_2 k$ уровней рекурсии и выполнение $k-1$ деления пополам. В случае когда требуемое количество разбиений k не является степенью двойки, каждое деление пополам необходимо осуществлять в соответствующем соотношении.

Геометрические методы

Геометрические методы выполняют разбиение сетей, основываясь исключительно на координатной информации об узлах сети. Так как эти методы не принимают во внимание информацию о связности элементов сети, они не могут явно привести к минимизации суммарного веса граничных ребер (в терминах *графа*, соответствующего сети). Для минимизации межпроцессорных коммуникаций геометрические методы оптимизируют некоторые вспомогательные показатели (например, длину границы между разделенными участками сети).

Обычно геометрические методы не требуют большого объема вычислений, однако качество их разбиения уступает методам, принимающим во внимание связность элементов сети.

Покоординатное разбиение

Покоординатное разбиение – это метод, основанный на рекурсивном делении пополам сети по наиболее длинной стороне. Общая схема выполнения метода состоит в следующем. Сначала вычисляются центры масс элементов сети. Полученные точки проектируются на ось, соответствующую наибольшей стороне разделяемой сети. Таким образом мы получаем упорядоченный список всех элементов сети. Делением списка пополам (возможно, в нужной пропорции) мы получаем требуемую бисекцию. Аналогичным способом полученные фрагменты разбиения рекурсивно делятся на нужное число частей.

Метод координатного вложенного разбиения работает очень быстро и требует небольшого количества оперативной памяти. Однако получаемое разбиение уступает по качеству более сложным и вычислительно трудоемким методам. Кроме того, в случае сложной структуры сети алгоритм может получать разбиение с несвязанными подсетями.

Комбинаторные методы

В отличие от геометрических методов, комбинаторные алгоритмы обычно оперируют не с сетью, а с *графом*, построенным для этой сети. Соответственно, в отличие от геометрических схем, комбинаторные методы не принимают во внимание информацию о близости расположения элементов сети друг относительно друга, руководствуясь только смежностью вершин *графа*. Комбинаторные методы обычно обеспечивают более сбалансированное разбиение и меньшее информационное взаимодействие полученных подсетей. Однако комбинаторные методы имеют тенденцию работать существенно дольше, чем их геометрические аналоги.

Деление с учетом связности

С самых общих позиций понятно, что при разделении *графа* информационная зависимость между разделенными подграфами будет меньше, если соседние вершины будут находиться в одном подграфе. *Алгоритм деления графов с учетом связности* пытается достичь этого, последовательно добавляя к формируемому подграфу соседей. На каждой итерации алгоритма происходит разделение *графа* на 2 части. Таким образом, разделение *графа* на требуемое число частей достигается путем рекурсивного применения алгоритма.

Общая схема алгоритма может быть описана при помощи следующего набора правил.

- Iteration = 0.
- Присвоение номера Iteration произвольной вершине *графа*.
- Присвоение ненумерованным соседям вершин с номером Iteration номера Iteration + 1.
- Iteration = Iteration + 1.
- Если еще есть неперенумерованные соседи, то переход на шаг 3.
- Разделение *графа* на 2 части в порядке нумерации.

Для минимизации информационных зависимостей имеет смысл в качестве начальной выбирать граничную вершину. Поиск такой вершины можно осуществить методом, близким к рассмотренной схеме. Так, начиная нумерацию из произвольной вершины, мы можем взять любую вершину с максимальным номером. Как нетрудно убедиться, она будет граничной.

Алгоритм Кернигана – Лина

В *алгоритме Кернигана – Лина* используется несколько иной подход для решения *проблемы оптимального разбиения графа* – предполагается, что некоторое начальное разбиение *графа* уже существует, затем имеющееся приближение улучшается в течение некоторого количества итераций. Применяемый способ улучшения в алгоритме Кернигана – Лина состоит в обмене вершинами между подмножествами имеющегося разбиения *графа*. Для формирования требуемого количества частей *графа* может быть использована, как и ранее, рекурсивная процедура деления пополам. Общая схема одной итерации алгоритма Кернигана – Лина может быть представлена следующим образом.

- Формирование множества пар вершин для перестановки. Из вершин, которые еще не были переставлены на данной итерации, формируются все возможные пары (в парах должно присутствовать по одной вершине из каждой части имеющегося разбиения *графа*).
- Построение новых вариантов разбиения *графа*. Каждая пара, подготовленная на шаге 1, поочередно используется для обмена вершин между частями имеющегося разбиения *графа* для получения множества новых вариантов деления.
- Выбор лучшего варианта разбиения *графа*. Для сформированного на шаге 2 множества новых делений *графа* выбирается лучший вариант. Этот вариант далее фиксируется как новое текущее разбиение *графа*, а соответствующая выбранному варианту пара вершин отмечается как использованная на текущей итерации алгоритма.
- Проверка использования всех вершин. При наличии в *графе* вершин, еще не использованных при перестановках, выполнение итерации алгоритма снова продолжается с шага 1. Если же перебор вершин *графа* завершен, далее следует шаг 5.
- Выбор наилучшего варианта разбиения *графа*. Среди всех разбиений *графа*, полученных на шаге 3 проведенных итераций, выбирается (и фиксируется) наилучший вариант разбиения *графа*.

Основная литература: 3

Дополнительная литература: 1

Контрольные вопросы:

1. В чем состоит *задача поиска всех кратчайших путей*?
2. Приведите общую схему *алгоритма Флойда*. Какова трудоемкость алгоритма? В чем состоит способ распараллеливания *алгоритма Флойда*?
3. В чем заключается *задача нахождения минимального охватывающего дерева*?
4. Приведите общую схему *алгоритма Прима*. Какова трудоемкость алгоритма? В чем состоит способ распараллеливания *алгоритма Прима*?
5. В чем отличие *геометрических и комбинаторных методов разделения графа*? Какие методы являются более предпочтительными? Почему?