Построение MST на мультипроцессоре с ccNUMA архитектурой

Зайцев Вадим, zaic101@gmail.com
Новосибирский государственный университет*

В данной работе рассматривается задача построения минимального остовного дерева (minimum spanning tree – MST) на мультипроцессоре с ccNUMA архитектурой. Задача построения MST формулируется следующим образом: дан взвешенный, неориентированный граф и требуется найти остовное дерево (максимальный по включению рёбер подграф, не имеющий циклов), в котором сумма весов рёбер будет минимальна. В случае, когда исходный граф связный, в итоге будет построено остовное дерево, если же в исходном графе несколько компонент, то результатом будет лес.

Минимальные остовные деревья имеют широкое практическое применение. Они используются при построении различных сетей: коммуникационных линий, компьютерных сетей, транспортных сетей. Так же они используются в кластеризации, сегментации при обработке изображений, распознавании рукописного ввода.

Существует несколько различных последовательных алгоритмов построения MST, наиболее распространёнными из которых являются алгоритмы Крускала, Прима и Борувки [1]. Алгоритм Крускала состоит из шагов (на каждом шаге добавляется одно новое ребро в текующий остов), каждый следующий из которых зависит от результата предыдущего, что делает распараллеливание данного алгоритма неэффективным. В алгоритме Прима шаги так же связаны между собой, а распараллеливание одного шага не даст большого выигрыша в силу большого количества шагов и малой вычислительной сложности одного шага. Однако, существует параллельный алгоритм построения MST, являющийся модификацией алгоритма Прима. Алгоритм Борувки в своём оригинальном виде является наиболее пригодным для распараллеливания. Далее опишем алгоритм Борувки и модифицированный алгоритм Прима.

Алгоритм Борувки. Изначально каждая вершина графа считается отдельной компонентой, затем компоненты объединяются до тех пор, пока не останется ровно одна. Объединение компонент происходит итеративно: на каждой итерации для каждой компонеты просматривается список всех инцидентных ей рёбер и выбирается минимальное из них, затем компоненты объединяются по найденным рёбрам. За счёт того, что основная часть

*Работа поддержана Грантом Президента РФ для молодых учёных МК-3644.2014.9

алгоритма – обход всех рёбер, который может быть разбит на независимые части и выполнятся параллельно, данный алгоритм в перспективе может дать хорошую масштабируемость. В статьях [2, 3, 4] представлены различные варианты параллельных реализаций алгоритма Борувки. Разные реализации используют разные структуры данных для хранения рёбер (хранение рёбер одним большим списком или хранение для каждой вершины списка инцидентных ей рёбер) и разные способы их объединения (слияние и копирование списков, объединение списков за константное время и т. д.).

Модифицированный алгоритм Прима. В статье [3] представлен параллельный алгоритм построения MST на основе последовательного алгоритма Прима. Суть данного алгоритма в том, что каждый поток выбирает случайную вершину в графе и начинает "растить" дерево из выбранной вершины, пока не попытается присоединить вершину, уже принадлежащую другому дереву, после чего два дерева встретившихся потоков объединяются, один из них продолжает "растить" объединённое дерево, а второй заново выбирает вершину графа. К преимуществам данного подхода относится отсутствие барьерной синхронизации, которая требуется в алгоритме Борувки после каждой итерации и, в зависимости от реализации, между разными шагами одной итерации.

В данных работах [2, 3, 4] результаты производительности показаны либо для старых архитектур [2, 4], либо масштабируемость была ограничена шестью одноядерными процессорами (SMP UMA, UltraSPARC II) [3] .

Настоящая работа направлена на адаптацию существующих и реализацию новых алгоритмов, ориентированных на высокую эффективность на современных вычислителях с общей памятью с ccNUMA архитектурой.

Первая параллельная реализация построения MST основана на алгоритме Борувки. Изначально множество вершин графа разделяется между потоками равномерно по количеству инцидентных рёбер. Далее на каждой итерации выполняются следующие шаги:

1. *Минимальное инцидентное ребро (1).* Каждый поток просматривает своё подмножество рёбер и для каждой компоненты находит инцидентное ребро минимального веса. В результате каждый поток будет иметь массив, содержащий минимальное ребро для каждой компоненты среди просмотренного подмножества.

2. *Минимальное инцидентное ребро (2).* Происходит редукция: для каждой компоненты находится ребро минимального веса по всем потокам. Таким образом, для каждой компоненты определяется минимальное ребро уже по всему графу.

3. *Объединение деревьев (1).* Для каждой компонеты определяется её номер на следующей итерации. Так же решается проблема с возможными циклами (когда более двух компонент по кругу выберут следующую в качестве ближайшей), которые возможны в алгоритме Борувки при наличии рёбер одинакового веса.

4. *Объединение деревьев (2).* Осуществляется перенумерация компонент: для каждой вершины вычисляется номер компоненты, в которую она входит, используя параллельный алгоритм Pointer Jumping [5].

Каждый шаг исполняется параллельно всеми потоками, однако после каждого шага требуется барьерная синхронизация всех потоков. Далее были сделаны следующие оптимизации:

- Сортировка рёбер по весу и удаление петель, за счёт чего в среднем сокращается количество рёбер, которые необходимо просмотреть на первом шаге.
- Выделение и инициализация памяти с ориентацией на NUMA архитектуру.
- Иерархическая редукция по дереву на втором шаге с учётом NUMA архитектуры.

Наибольшая вычислительная сложность первого алгоритма заключается в первом шаге, в котором необходимо обойти бо́льшую часть рёбер графа, однако, основным препятствием для хорошей масштабируемости является то, что размер массива, редуцируемого на втором шаге, увеличивается с ростом количества потоков.

Вторая реализация алгоритма Борувки для хранения рёбер использует списки смежности. Изначально для каждой вершины сортируется список инцидентных ей рёбер по увеличению веса. Таким образом, получить самое лёгкое ребро, инцидентное вершине, возможно за $O(1)$, не выполняя проход по массиву. Далее выполняются следующие шаги, пока не останется одна компонента:

1. *Минимальное инцидентное ребро.* Для каждой компоненты берётся инцидентное ей ребро минимального веса. Как отмечено выше, это будет первое ребро из списка смежности.
2. *Объединение деревьев.* Объединение компонент, для каждой компоненты определяется её номер на следующей итерации. Аналогично шагам *Объединение деревьев 1-2* в первом алгоритме.
3. *Слияние списков.* Происходит объединение компонент путём слияния списка рёбер. Поскольку, списки уже отсортированы по возрастанию веса, то возможно слияние, при котором в результате так же будет получен отсортированный список, с сохранением линейного времени работы.
4. *Перенумерация.* Обход списков рёбер и перенумерация компонент: старые номера заменяются на новые, полученные на шаге *Объединения деревьев*.

В ходе анализа работы шага *Объединение деревьев* на тестируемы типах графов было выяснено, что количество объединяемых компонент в одну в большинстве случаев не преводсходит трёх и это позволяет быстро и эффективно отсеивать петли на шаге *Слияния списков* за счёт сокращения нерегулярных обращений в память.

Тестирование проводилась на графах, состоящих из $10^6 - 10^8$ вершин и $10^7 - 10^9$ рёбер: это RMAT-графы степени $20 - 23$, двумерные и трёхмерные решётки, а так же случайные графы.

Характеристики систем, на которых производились запуски:

- 2 × Intel Xeon CPU E5-2690 (8 ядер 2.9 GHz, 32KB L1 cache, 256KB L2 cache, 20MB L3 cache)
- 8 × Intel Xeon CPU E7-4870 (10 ядер 2.4 GHz, 32KB L1 cache, 256KB L2 cache, 30MB L3 cache)

Список литературы

[1] Т. Кормен. *Алгоритмы: построение и анализ.* Вильямс, 2005.

[2] Silvia Gotz Frank Dehne. Practical parallel algorithms for minimum spanning trees. *http://people.scs.carleton.ca/~dehne/publications/2-47.pdf.*

[3] Guojing Cong David A. Bader. Fast shared-memory algorithms for computing the minimum spanning forest of sparse graphs. *http://www.cc.gatech.edu/~bader/papers/MST-JPDC.pdf.*

[4] Anne Condon Sun Chung. Parallel implementation of boruvka minimum spanning tree algorithm. *http://minds.wisconsin.edu/bitstream/handle/1793/60020/TR1297.pdf.*

[5] Guy E. Blelloch and Bruce M. Maggs. Parallel algorithms. *http://homes.cs.washington.edu/~arvind/cs424/readings/BlellochM96b.pdf.*