Московский государственный университет имени М.В.Ломоносова

ФИЗИЧЕСКИЙ ФАКУЛЬТЕТ

Курсовая работа по дисциплине “Параллельное программирование”

Образование гигантского кластера в случайных сетях.

Выполнил студент 2 курса группы 214
Котелевский Никита Юрьевич

Научный руководитель
Тамм Михаил Владимирович

Содержание

Аннотация

В работе рассматривается эволюция сложной сети, в которой образуются связи между вершинами - ребра. Осуществляется поиск кластеров – связных компонент сложной сети. Получается распределение числа кластеров по времени, по их диаметру и размеру, а также распределение средней степени вершин.

1 Введение

Неориентированным графом (далее граф) в математике принято называть упорядоченную пару $G : (V, E)$, где V - это непустое множество вершин (узлов), а E – множество неупорядоченных пар вершин, называемых рёбрами.
Взвешенным графом будем называть граф, каждому ребру которого поставлено в соответствие некое значение (вес ребра).
Связный граф – граф, в котором все вершины связаны, то есть существует хотя бы один путь из одной вершины в другую.
Кластер – связный подграф.
Степень вершины - число связей, которые она образует.
Сложные (большие) сети– это графы, содержащие большое число вершин.
Сегодня сложные сети находят свое применение не только в областях естественных, но также и социальных наук. Так, например:

- Социальные сети
- Компьютерные сети, включая интернет
- Энергетические сети
- Нейронные сети мозга

Столь широкая область приложения сложных сетей и объясняет актуальность выбранной темы работы.

2 Численные методы

В данной задаче для подсчета диаметров кластеров используется алгоритм Флойда - алгоритм, позволяющий найти длины кратчайших путей между всеми парами вершин во взвешенном неориентированном графе. Для хранения графа используется матрица смежности. Если ребро существует, то на пересечении i-ой строчки и j-го столбца стоит 1, если же ребра нет, то на пересечении стоит отрицательное число -1. Диагональные элементы равны 0. Последовательный алгоритм Флойда имеет сложность n^3. Основная вычислительная нагрузка при решении задачи поиска кратчайших путей состоит в выполнении операции выбора минимального значения. Данная операция является достаточно простой и ее распараллеливание не приведет к заметному ускорению вычислений [2]. Более эффективный способ организации параллельных вычислений может состоять в одновременном выполнении нескольких операций обновления значений матрицы A.

С этой целью будем использовать ленточную схему разбиения матрицы A. Распараллеливание будет осуществляться с использованием технологии OpenMP.

3 Реализация

Определим классы `cluster` - кластеров и `vertex` - вершин, в которых имеется полезная информация об этих объектах. Так, класс вершин позволяет знать степень вершины v, номер кластера *incluster*, к которому она относится, и массив связей *connections* размера v, содержащий номера вершин, с которыми соединена данная. Класс кластеров содержит информацию о его размере *size*, массив *top* размера *size*, содержащий номера вершин, принадлежащих кластеру, а так же диаметр *diametr* кластера.

В данной задаче рассматривается сеть, состоящая из $N \gg 1$ узлов и её эволюция в течение времени T с шагом по времени $\frac{T}{rr}$, где значения rr, T, N и tt (момент времени, в который рассматривается распределение вершин по степеням) задаёт пользователь. Распределение вершин по степеням в фиксированный момент времени, в соответствии с теорией [1] для данной модели, подчиняется распределению Пуассона $n_i(t) = \frac{t^i}{i!}e^{-t}$, где $n_i(t)$ - доля вершин степени i в момент времени t. Первоначально связей нет. Задана вероятность p=$\frac{1}{N-1} \ll 1$ образования связи между двумя узлами, которая проводится только в том случае, когда её ещё нет. Случайность проведения связи осуществляется путем сравнения псевдослучайного числа, заключенного от 0 до 1, с вероятностью p. Если это случайное число больше p и связи между вершинами ещё нет, то ребро проводится, если же что-то из этих условий не выполнено – связь не проводится. Проверка существования ребра осуществляется с помощью функции класса `vertex` *connect*(), которая возвращает значение *true* в том случае, когда проводим связь. Сразу запоминаем образовавшиеся при рассмотрении вершины связи и помещаем их в соответствующий массив *connections* в классе `vertex`.

Поскольку целью задачи является сбор статистики и усреднение по числу реализаций *Rest*, которое так же задаётся пользователем, то целесообразно разделить число реализаций между процессами. В данной задаче это осуществляется с помощью технологии MPI: в мастер процессе (процессе с рангом 0) осуществляется подсчет числа реализаций для каждого процесса (учтено, что число реализаций может нацело не делиться на число процессов) и рассылка этого числа соответствующим процессам.

После получения каждым потоком всех данных, необходимо обновить в каждом процессе генерацию случайных чисел, для чего используем srand(time(NULL)+myrank), (где myrank - ранг текущего процесса) чтобы в каждом из процессов не генерировалась одна и та же последовательность чисел. Далее в каждом процессе запускается цикл по чис-

лу повторений, в котором осуществляется обход всех вершин в каждый момент времени и регистрация образовавшихся связей. В случае образования связи между двумя вершинами, увеличиваем их степень на 1 и добавляем их в массив связей друг у друга.

Далее идет рассмотрение кластеров: если связь соединяет вершины из разных кластеров (первоначально число кластеров равно числу вершин и каждый имеет порядковый номер, соответствующий номеру содержащейся в нём вершины), то кластеры объединяются, и результирующему кластеру приписывается наименьший из первых двух порядковый номер. Поиск диаметра кластера (наибольшее расстояние между двумя вершинами) осуществляется с помощью алгоритма Флойда, который, в силу того, что он расположен внутри процесса, распараллеливается с помощью OpenMP.

Далее осуществляется поиск максимального и второго по размерам кластеров. Фиксируем также и их диаметры.

Каждая величина усредняется по общему числу реализаций. После завершения повторений в каждом процессе, получившиеся значения отправляются в процесс с рангом 0, в котором они суммируются. Таким образом получается, что усреднение произошло по числу $Rest$, заявленных пользователем.

На выход программы подаются следующие графики:

- Средняя степень вершин от времени
- Распределение степени вершин в фиксированный момент времени
- Число кластеров от времени
- Диаметр максимального и второго по размерам кластеров от времени
- Размер максимального и второго по размерам кластеров от времени

Условно работу программы можно представить в виде следующей блок-схемы:

Рис. 1: Блок-схема программы.

4 Результаты

В качестве результатов получены следующие графики (на примере 10000 реализаций для 100 вершин, в течение 10-ти секунд с шагом по времени 0,1).

- Рассмотрим среднюю степень вершин. По определению это

  $< k >= \frac{\sum_{i=1}^{k} k_i}{k}$, где k - число всех вершин, а k_i - степень i-ой вершины.

  $\frac{\sum_{i=1}^{k} k_i}{k} = \frac{1}{k}\frac{2t}{k-1}\frac{k(k-1)}{2} = t$, так как сумма всех степеней вершин это удвоенное количество имеющихся в системе связей. То есть

  $< k >= t$.

- Распределение вершин по степеням подчиняется следующему дифференциальному уравнению(где $n_k = \frac{N_k}{k}$ - доля вершин степени k):

  $\frac{dn_k}{dt} = n_{k-1} - n_k$

  Решение которого имеет вид:

  $n_k(t) = \frac{t^k}{k!}e^{-t}$

- В графе Эрдёша-Реньи в момент времени $t = 1$ происходит перколяционный переход, в результате которого образуется гигантская компонента, в конечном итоге поглощающая все остальные кластеры.

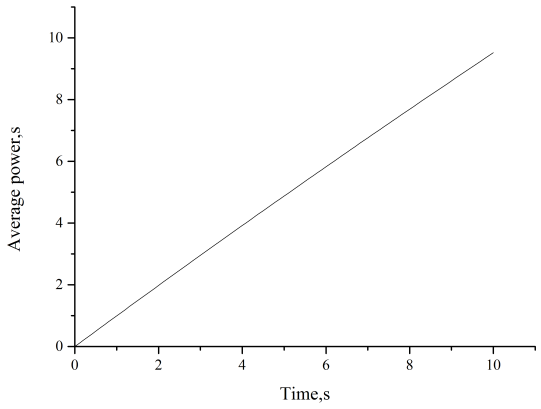

Рис. 2: Средняя степень вершин с течением времени.

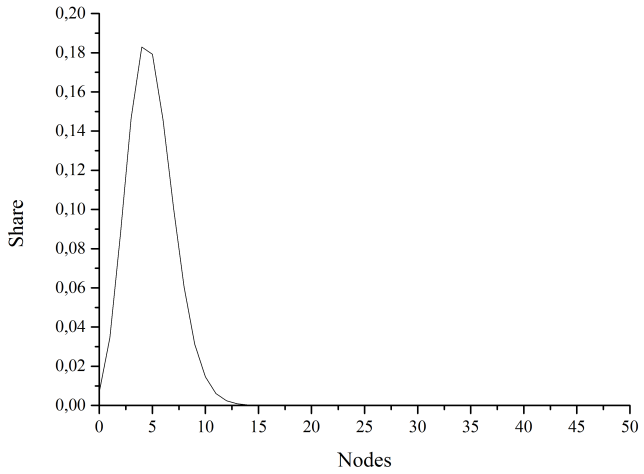

Рис. 3: Доли вершин разных степеней в фиксированный момент времени.

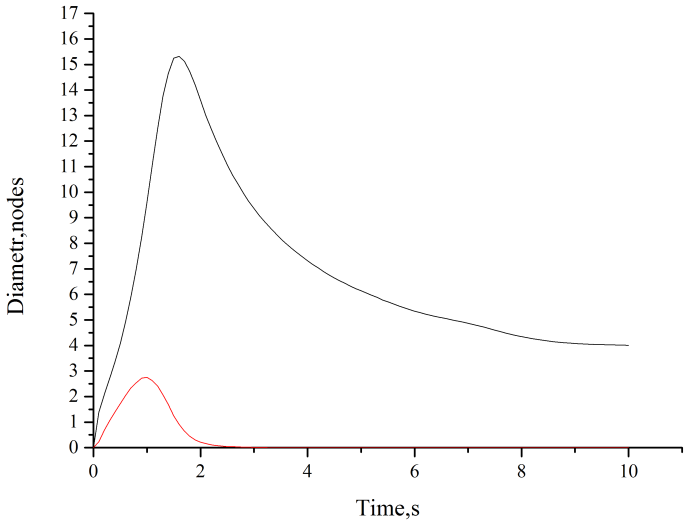

Рис. 4: Диаметры максимального по размерам и второго кластеров.

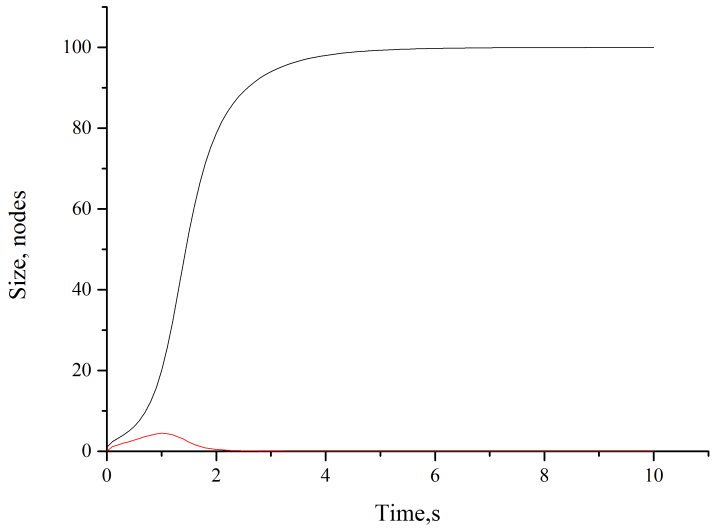

Рис. 5: Размеры максимального и второго по размерам кластеров.

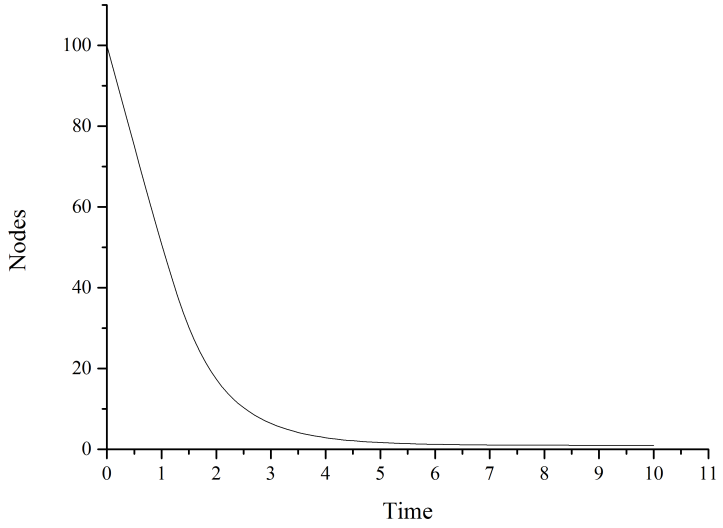

Рис. 6: Число кластеров с течением времени.

5 Итоги

В результате выполнения работы получены программы, которые с помощью MPI и OpenMP/MPI распараллеливают исходную программу.

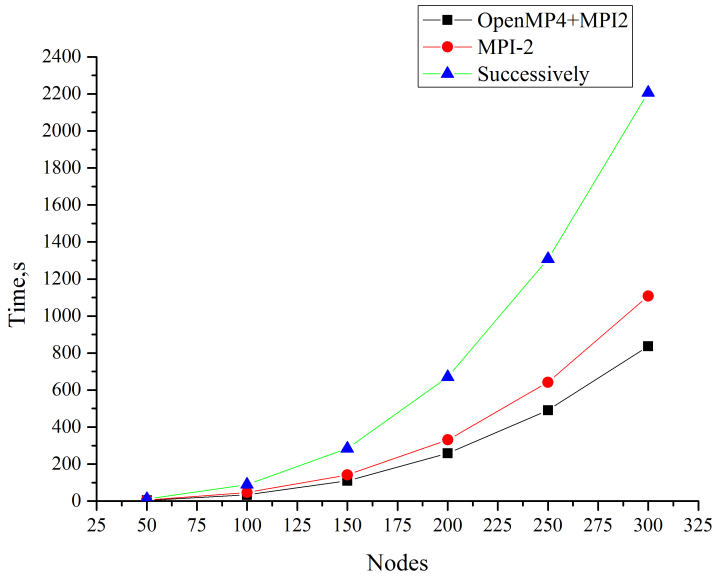

Рис. 7: Зависимость времени работы трёх алгоритмов (последовательный, параллельный с использованием только MPI (на 2х процессах) и MPI (на 2х процессах)+OpenMP (4 нити)) от числа вершин.

В соответствии с законом Амдала $T_1 = T_s + T_p$ - один процесс. $T_N = T_s + \frac{T_p}{N}$ - N процессов. Коэффициент ускорения программы $K = \frac{T_1}{T_N}$ равен:

$K = 2,62$ для алгоритма, использующего в каждой MPI ветви OpenMP.

$K = 1,98$ для алгоритма, использующего только MPI.

Таким образом, использование OpenMP в данной задаче целесообразно, что наглядно видно из графика: алгоритм Флойда, сложность которого составляет n^3 (где n - число узлов), с использованием OpenMP понижается. Из графика видно, что время выполнения программы уменьшается с ростом числа процессов. Ускорения, достигнутые на разном числе процессов, равны:

$K_2 = 1,99$

$K_4 = 3,69$

$K_8 = 7,93$

$K_{16} = 15,82$

$K_{32} = 31,26$

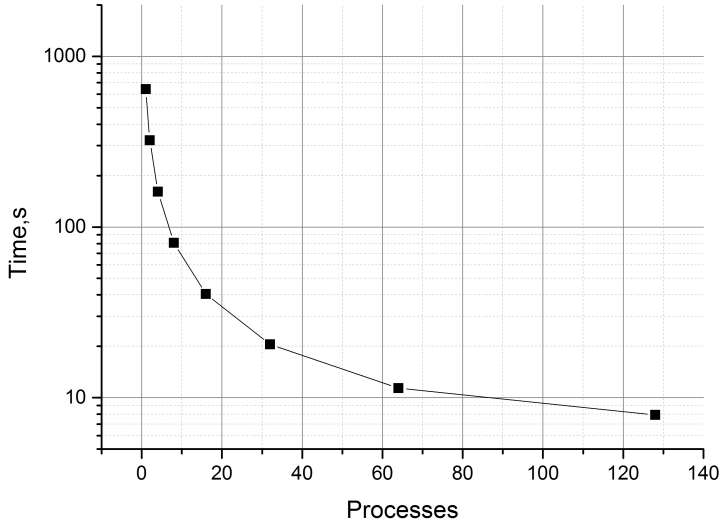

Рис. 8: Зависимости времени выполнения программы, использующей только MPI, от числа процессов со следующими стартовыми параметрами: число узлов 100, промежуток времени 10 секунд с шагом 0.1 секунда, число повторений - 700 раз.

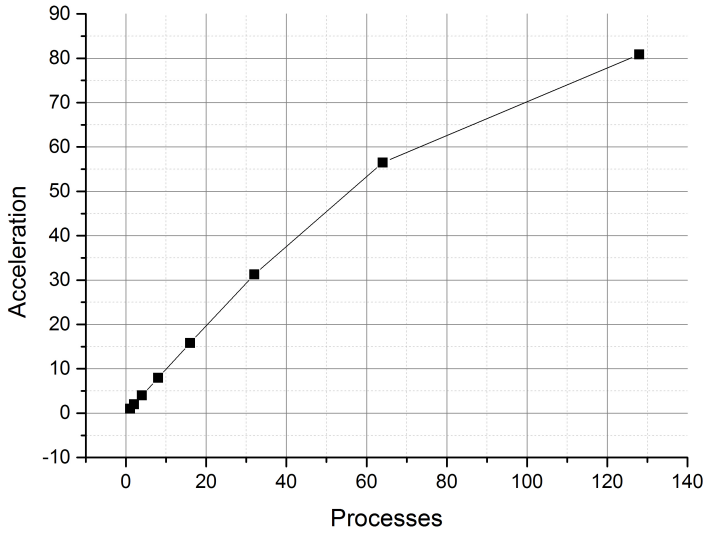

Рис. 9: График зависимости ускорения от числа процессов.

$K_{64} = 56,5$
$K_{128} = 80,86$

Из закона Амдала $K = \frac{1}{S+\frac{P}{N}}$, где S - доля последовательной части, P - доля параллельной части, N - число процессов, следует, что доля параллельной части программы составляет $99,5\%$ Таким образом, применение методов параллельного программирования целесообразно для ускорения данной задачи.

6 Литература

1 Кинетический взгляд на статистическую физику, Павел Л. Крапивский, Сидней Реднер, Эли Бен-Наим. Глава 14 "Сложные сети".

2 Параллельные алгоритмы на графах. Лекция 10. Гергель В.П.

3 Глоссарий теории графов, wikipedia.com

4 Работа выполнена с использованием ресурсов суперкомпьютерного комплекса МГУ имени М.В. Ломоносова. Воеводин Вл.В., Жуматий С.А., Соболев С.И., Антонов А.С., Брызгалов П.А., Никитенко Д.А., Стефанов К.С., Воеводин Вад.В. Практика суперкомпьютера "Ломоносов"// Открытые системы. - Москва: Издательский дом "Открытые системы N 7, 2012. С. 36-39.