МОДЕЛИРОВАНИЕ РАСПАРАЛЛЕЛИВАНИЯ ЗАДАЧИ С ПОМОЩЬЮ ГРАФА ПОТОКОВ ДАННЫХ. ОРГАНИЗАЦИЯ ДАННЫХ

Костин В. И., Краснокутская М. В.
Донецкий Национальный Технический Университет, ФВТИ, ПМИ
krasnokutskaya@maus.donetsk.ua

Abstract

Kostin V.I., Krasnokutskaya M.V. *Simulation of problem parallelization by means of dataflow graph. Data organization. We describe a flowgraph representation of a problem parallelization. Balancing of the computational load across processors is abstracted to a graph partitioning problem. We propose review algorithms to solve this problem and describe some peculiarities of their use to graphs with high number of nodes.*

Введение

Параллельные вычисления позволяют значительно повысить эффективность и скорость обработки информации при решении современных задач. Такие задачи возникают в электромеханике, при оптимизации сложных систем, при прогнозировании погоды, моделировании разнообразных технических и природных процессов.

Одна из основных проблем, которая возникает в каждом параллельном вычислении, это распределение обработки данных между процессорами. Решением может быть использование математической модели, в основе которой лежит граф потоков данных (ГПД). Программа представляется набором вычислительных узлов-подзадач, которые имеют фиксированное количество информационных входов и выходов. Каждая подзадача выполняется на отдельном процессоре. Узлы-подзадачи – вершины графа потоков данных, а информационные потоки между ними – ребра графа. Оптимальное распределение обработки данных между процессорами минимизирует время выполнения всех вычислений. Задача распределения обработки данных на процессоры сводится к задаче разбиения графа. Необходимо разбить граф потоков данных так, чтобы количество связей между подграфами было минимальным. Практический опыт показал, что качество распределения задач между процессорами

сильно влияет на производительность, что обусловило значительный интерес к алгоритмам разбиения графов [1].

К сожалению, разбиение графа – NP-сложная задача, и поэтому все известные алгоритмы разбиения являются эвристическими и дают приближенный к оптимальному результат. Однако, не смотря на это ограничение, было разработано много алгоритмов разбиения графов, дающих высококачественное разбиение за малое время [2].

При программной реализации любого из этих алгоритмов возникает задача выбора типа данных для представления информации о графе. Существуют различные способы внутреннего представления графов в оперативной памяти ЭВМ, в том числе в виде списков (массивов) вершин и ребер, списков (массивов) смежности, матриц смежности, а также в виде комбинаций этих структур хранения. Выбор внутреннего представления оказывает решающее влияние на эффективность выполнения различных операций над графами [3].

Целью данной работы является исследование методов организации данных в задачах разбиения графов больших размерностей.

1. Постановка задачи о разбиении графа

Исходным объектом для задач разбиения служит неориентированный граф. Пространством решений служит множество всевозможных разбиений графа на непересекающиеся подграфы. На разбиения графа могут накладываться ограничения D. Для различных задач, естественно, возможны различные ограничения.

Задачи декомпозиции имеют следующий набор основных критериев:

• число внешних соединений между подграфами;

• число подграфов.

Первый критерий определяется функционалом внешних связей. Смысл второго критерия очевиден: он просто равен числу подграфов разбиения. Постановки задач могут варьироваться в зависимости от свойств графа, которые задаются моделируемым объектом.

Пусть задан неориентированный взвешенный граф G(V,E) порядка n, где $V=\{v_1,\ldots,v_n\}$ – множество вершин; $E \subseteq V \times V$ – множество ребер.

Требуется определить разбиение множества вершин V графа G(V,E) на k – подмножеств $(V_1,\ldots,V_k)$ таким образом, чтобы для частей графа $G1(V_1,E_1),\ldots, Gk(V_k,E_k)$ выполнялись следующие требования:

$$V_i \cap V_j = \varnothing, \text{ для } \forall\, i \neq j, \text{ где } i,j = \overline{1,k};$$

$$\bigcup_{i=1}^{k} V_i = V \qquad (1.1)$$

$$|V_1| = n_1, \ldots, |V_k| = n_k,\ n_1 + \ldots + n_k = n,$$

Сечением разбиения $C(V_1,\ldots,V_k)$ будем называть совокупность ребер, соединяющих вершины, которые принадлежат разным подграфам.

В качестве критерия оптимальности Q, определяющего эффективность биразбиения (бисекции) $(V_1,\ldots,V_k)$ будем рассматривать вес сечения - сумма всех ребер сечения:

$$Q(V_1, V_2, \ldots, V_k) = \frac{1}{2}\sum_{L=1}^{k-1}\sum_{i\in E_L}\sum_{j\notin E_L} 1 \rightarrow \mathbf{min} \qquad (1.2)$$

В этом случае оптимальным k-разбиением является решение $(V_1^*,\ldots,V_k^*)$ экстремальной задачи (1.2), то есть разбиение $(V_1^*,\ldots,V_k^*)$ с минимальным весом сечения $C(V_1^*,\ldots,V_k^*)$.

Частным случаем задачи k-разбиения является задача (1.1) бисекции графа – когда граф разбивается на два подграфа (k=2). В этом случае решение задачи разбиения графа (V_1,V_2) будем называть бисекцией.

Система требований (1.2), предъявленных к разбиению $(V_1,\ldots,V_k)$, определяет область поиска – D – задачи разбиения графа. Данная задача относится к задачам переборного типа и общее число допустимых решений |D| можно вычислить из выражения:

$$\frac{n!}{n_1!\cdot n_2!\cdot\ldots\cdot n_k!\cdot t!} \qquad (1.3)$$

где t – общее число подграфов, имеющих одинаковые размерности [4].

2. Обзор алгоритмов разбиения графов

Среди наиболее известных алгоритмов разбиения графов можно выделить алгоритмы:

- алгоритм Kernighan-Lin / Fiduccia-Mattheyses (KL/FM);
- уровневое ячеечное разбиение;
- многоуровневые схемы;
- алгоритм спектральной бисекции.

Kernighan-Lin алгоритм (KL) разработан в 60-х годах прошлого столетия. Алгоритм используется для улучшения начального (возможно случайного) разбиения графа, путем обмена вершинами из начальных наборов с целью уменьшения числа разрезающих ребер. KL алгоритм основан на понятии веса - величины, которая определяет выигрыш от перемещения вершины из одного подмножества в другое. Вес рассчитывается для каждой вершины как количество соединений вершины с другим подмножеством, минус количество соединений с подмножеством, в котором вершина находится. Пока есть вершины с положительным весом, алгоритм меняет вершины с максимальным весом местами с вершинами из другого подмножества.

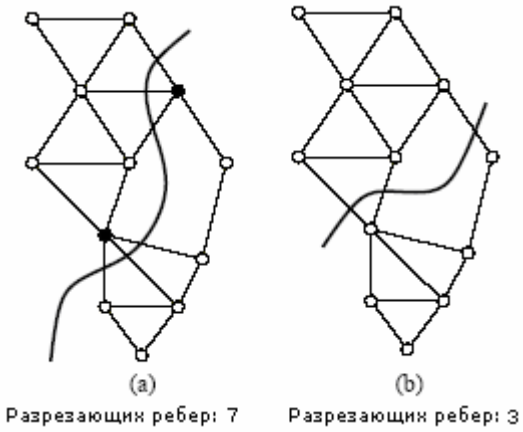
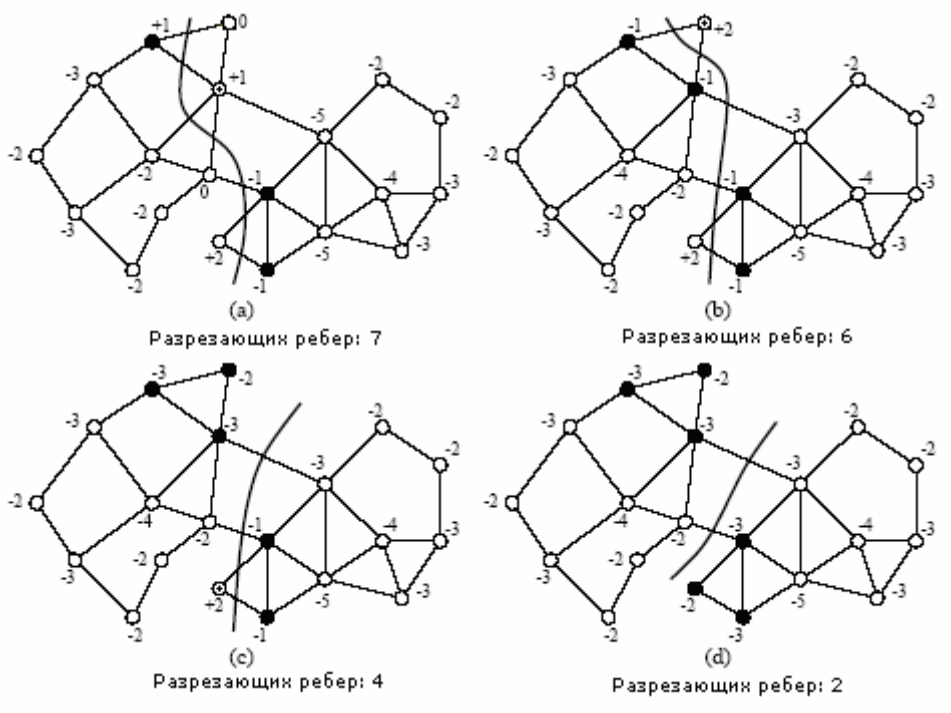


Рисунок 1. Шаг KL – алгоритма

Недостатком данного подхода является то, что перемещения вершин могут привести к локальному минимуму. Для преодоления локальных минимумов алгоритм использует поиск экстремума – вершины с отрицательным весом перемещаются с целью нахождения более глобального минимума. Причем, если вершина уже была перемещена, то она не участвует в дальнейшем просмотре. Во время этих перемещений алгоритм запоминает лучшее полученное разбиение. Когда перемещения закончены, алгоритм восстанавливает это наилучшее разбиение. Этот процесс может повторяться, используя новое разбиение как стартовую точку [5].

Fiduccia and Mattheyses предложили модификацию KL алгоритма. FM алгоритм основан на KL-алгоритме, но за один его шаг перемещается только одна вершина, после этого для каждой вершины пересчитывается вес. Затем для каждого подмножества строится очередь вершин. Вершины помещаются в нее по убывания веса. Просмотр вершин происходит в том порядке, в котором вершины расположены в очереди. Если условия балансировки нарушается, вершина не может быть перенесена. При перемещении вершины в другой набор она удаляется из очереди. При пересчете весов всех вершин очереди строятся заново.

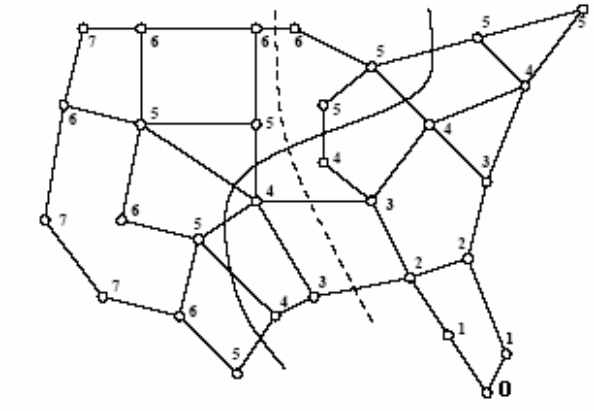


Шаги KL/FM-алгоритма

Белым помечена вершина, которая перемещается, черным - очереди вершин

Рисунок 2. Шаги KL/FM-алгоритма

Уровневое ячеечное разбиение. Обычно разбиение будет иметь мало разрезающих ребер, если смежные вершины находятся в одном подмножестве. Уровневое ячеечное разбиение (Levelized Nested Dissection - LND) пытается помещать связанные вершины вместе, начинаясь с подмножества, которое содержит только одну вершину и затем увеличивает подмножество путем добавления смежных вершин. Алгоритм заключается в следующем: выбирается начальная вершина и помечается номером 0. Потом все смежные с ней вершины помечаются номером 1. Затем всем вершинам, которые еще не помечены и связанным с уже помеченным номером вершинам назначается этот номер, увеличенный на 1 и т.д. Этот процесс продолжается до тех пор, пока половине вершин не будут назначены номера. Помеченные вершины образуют одно подмножество, а непомеченные вершины – другое.



Сплошной линией показано разбиение, полученное LND-алгоритмом.

Пунктирная линия показывает разбиение более высокого качества

Рисунок 3. LND-разбиение

К недостатку данного алгоритма можно отнести то, что полученное таким образом разбиение сильно зависит от начальной выбранной вершины. Поэтому можно сделать вывод, что данный алгоритм должен быть дополнен методами отыскания начальной вершины.

Многоуровневые схемы. Этот класс алгоритмов разбиения основывается на многоуровневом подходе. Алгоритм состоит из трех фаз: стадия укрупнения, стадия разбиения, стадия многоуровневого улучшения.

На стадии укрупнения графа формируется последовательность графов, каждый из которых получается из предыдущего путем объединения смежных вершин, пока не получится наименьший граф. Затем производится начальное разбиение наименьшего графа. После этого разбиение улучшается на графе каждого уровня, начиная с минимального и заканчивая исходным графом (для этого может использоваться KL/FM алгоритм или алгоритм спектральной бисекции).

Многоуровневые алгоритмы эффективны по двум причинам. Во-первых, алгоритм позволяет скрыть большое количество ребер на укрупненном графе. Во-вторых, KL/FM метод работает более эффективно при многоуровневой схеме. Перемещение одной вершины в укрупненном графе эквивалентно перемещению большого числа вершин исходного графа. Перемещение группы вершин позволяет избежать некоторых видов локальных минимумов.

Алгоритм спектральной бисекции [1]. Алгоритм спектральной бисекции основан на использовании собственных векторов и чисел. Этот метод получил большое внимание, так как дает хорошее соотношение между универсальностью, качеством и эффективностью.

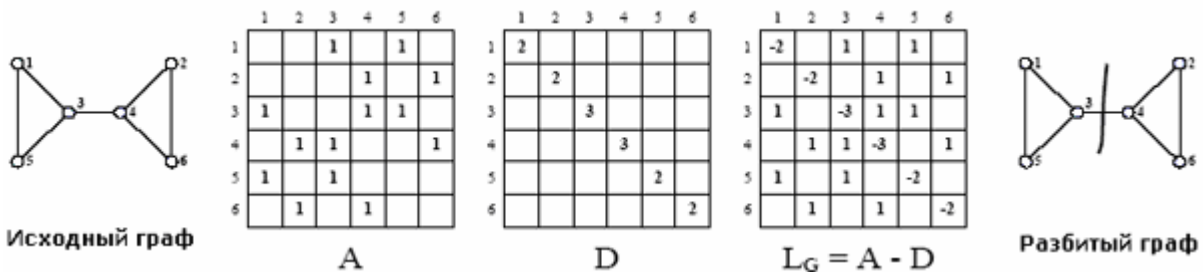


Рисунок 4. Разбиение графа методом спектральной бисекции

На рисунке изображено разбиение графа методом спектральной бисекции. По этому алгоритму в соответствии каждой вершине графа ставится переменная x, равная +1 или –1, таким образом, что сумма всех x-ов равна 0. Первое условие подразумевает разбиение на два различных набора. Второе требует чтобы наборы были равного размера, учитывая четность исходного количества. Назовем вектор x вектор-индикатор, так как он показывает принадлежность каждой вершины к набору.

Затем определяется функция от x, определяющая число граней, пересекающихся между наборами. Эта функция для минимизации преобразуется к матричной форме, используя матрицу Лапласа L.

Минимизировать $\frac{1}{4} x^T L x$ (2.1)

При условии: $x^T I = 0, \quad x_i = \pm 1$

Так как разбиение графа – NP-трудная задача, необходимо ослабить ограничения дискретности на x и сформулировать новую непрерывную задачу. Эта непрерывная задача – только приближение к дискретной, и значения, определяющие ее решение, должны быть отображены обратно к ±1 в соответствии с некоторой соответствующей схемой. Идеально, когда решение близко к ±1 .

Минимизировать $\frac{1}{4} x^T L x$ (2.1)

При условии: $x^T I = 0, \quad x^T x = n$

Если U_1, U_2 …– нормализованные собственные векторы L с соответствующими собственными значениями $\lambda_1 <= \lambda_2 <= \lambda_3$…., то матрица L имеет следующие свойства:

(I) L симметрична.

(II) Ui попарно ортогональны.

(III) U1=n-0.51, λ1=0

(IV) Если граф замкнутый, то только λ1 принимает нулевое значение.

Затем выразим X в терминах собственных векторов L: $x=\Sigma\alpha_i U_i$, где α_i – вещественные константы, такие, что $\Sigma(\alpha_i)^2 = n$. Свойство (II) гарантирует, что это всегда возможно. Заменой x мы получаем функцию для минимизации, зависящую от собственного значения матрицы Лапласа λ_2. $f(x)=0.25(\alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \ldots + \alpha_n^2 \lambda_n)$, начиная с $\lambda_1=0$. Очевидно

$(\alpha_2^2 + \alpha_3^2 + \ldots + \alpha_n^2) \lambda_2 <= (\alpha_2^2 \lambda_2 + \alpha_3^2 \lambda_3 + \ldots + \alpha_n^2 \lambda_n)$ учитывая упорядоченность собственных величин $f(x)>=n \lambda_2/4$.

Мы можем минимизировать $f(x)=n \lambda_2/4$, выбирая $x = \sqrt{n} U_2$.

Полученный вектор x – решение непрерывной задачи. Остается решить задачу отображения вектора x к дискретному разделению. Для этого находится медиана значений x_i, и затем отображаются вершины выше значения медианы в один набор, ниже - в другой. Если несколько вершин имеют значение медианы, то они распределяются не нарушая равновесия. Это решение - самая близкая дискретная точка к непрерывному оптимуму.

3 Исследование методов организации данных

Как уже говорилось, при реализации алгоритма встает задача выбора типа данных для представления информации о графе.

Задание графов с помощью матриц удобно для алгоритмов, использующие матричные вычисления (например, алгоритм спектральной бисекции). Однако, при обработке графа большой размерности (n=1000, 10000), матрицы занимают слишком много памяти. Следует учесть, что матрицы графов потоков данных довольно разрежены, то есть матрицы содержат много нулей.

В качестве предварительного исследования рассматривалось умножение матрицы Лапласа на вектор. Рассмотрим следующие представления матрицы при программной реализации алгоритма:

- двумерный массив;
- массив динамических массивов (первый элемент динамического массива – число ненулевых элементов в строке матрицы);
- три массива: массив ненулевых элементов матрицы, массивы индексов строк и столбцов, соответствующих этим элементам;
- матрица в формате RR(C)O. Сокращенное название данного формата происходит от английского словосочетания "Row - wise Representation Complete and Ordered" (строчное представление, полное и упорядоченное) [6].

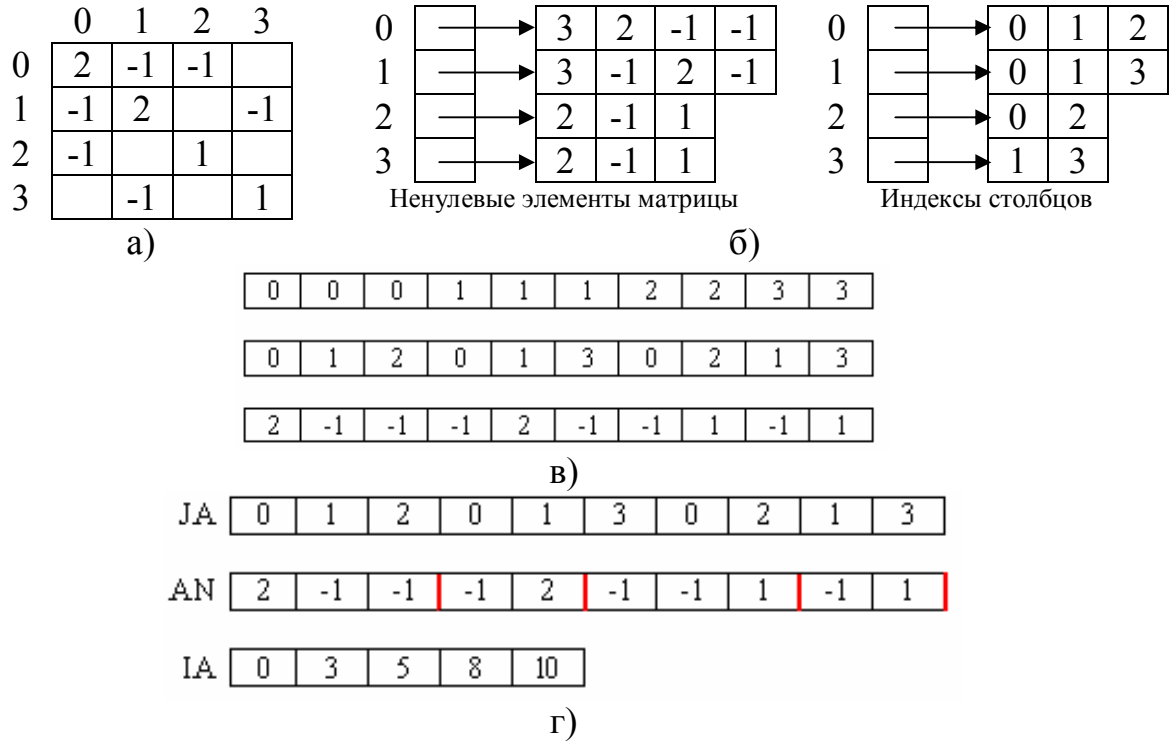


Рисунок 5 – Способы представления матрицы: а) двумерный массив; б) массив динамических массивов; в) три массива; г) матрица в формате RR(C)O

Представляя матрицу Лапласа двумерным массивом, мы занимаем n^2 единиц памяти. Массив динамических массивов занимает 2n+m единиц

памяти, где – число ребер графа. Три динамических массива – 3m +n единицы памяти. Матрица в формате RR(C) – 2m +n+1 единицы.

На рисунке 5 приведены результаты предварительных исследований, диаграммы зависимости времени умножения матриц от степени разреженности матрицы для матрицы размерностью 1000×1000. Для вычисления каждой точки графика задавалась размерность матрицы, число ненулевых элементов. По этим данным случайным образом создавалось 10 матриц. Каждая матрица умножалась на вектор 100 раз. Для построения диаграммы бралось среднее время умножения всех этих матриц на вектор. Вычисления проводились на компьютере с процессором Intel Celeron с тактовой частотой 2,93 GHz и оперативной памятью 1GB.

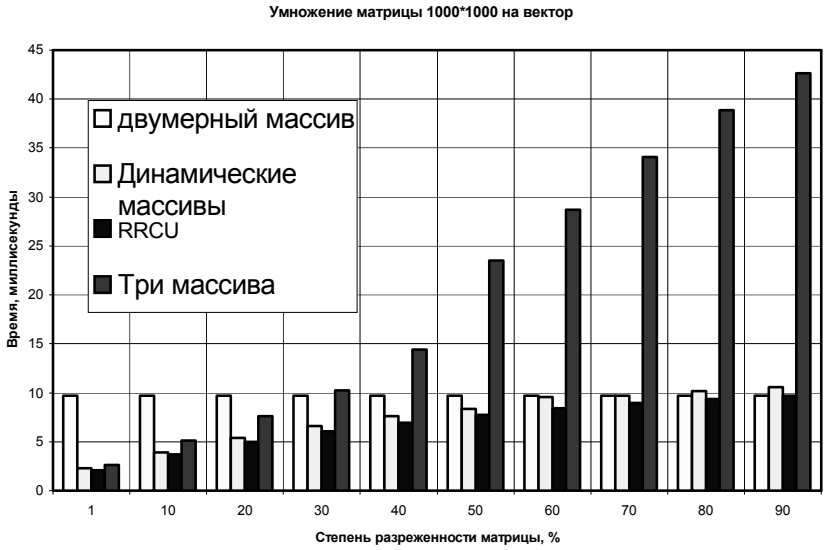


Рисунок 6. Время умножения матрицы 1000×1000 на вектор

Как видно, степень разреженности матрицы мало влияет на скорость умножения, когда матрица задана двумерным массивом. При реализации матрицы с помощью массива динамических массивов, трех массивов и в формате RR(C)O видно, что чем больше число ненулевых элементов в матрице, тем больше время умножения. Причем до определенного значения это время меньше соответствующего времени для двумерного массива (для матрицы, заданной динамическими массивами и в формате RR(C)O). Для матрицы, заданной тремя массивами это не выполняется. Время ее умножения на вектор больше времени умножения двумерного массива при любой степени разреженности матрицы.

Представление матрицы двумерным массивом проще в реализации. Объем занимаемой памяти и время умножения матрицы на вектор при такой реализации – постоянны. Такой способ оптимально подходит для неразреженных матриц с большим числом ненулевых элементов.

Разреженные матрицы лучше представлять массивом динамических массивов или в формате RR(C)O, так как при такой реализации объем занимаемой памяти и время умножения матрицы на вектор зависит от размерности и степени разреженности графа. В дальнейшем исследовании будет сравниваться использование двумерного массива и матрицы в формате RR(C)O в задачах разбиения графов.

Разработана программа, определяющая разбиения графа, используя метод спектральной бисекции. Были получены результаты для графов с числом вершин 1000. Случайным образом были сгенерированы графы с заданным числом вершин и различными степенями разреженности матрицы Лапласа (5%, 10%, …, 95%б 100%). Для каждого графа 10 раз находилось разбиение и определялось среднее время, затраченное на разбиение. На рисунке 7 показана диаграмма зависимости времени разбиения графа от степени разреженности его матрицы Лапласа (то есть от количества ребер). Данные приведены для двух случаев: когда при разбиении для представления матрицы Лапласа использовался двумерный массив (матрица) и формат RR(C).

Вычисления проводились на компьютере с процессором Intel Pentium 4 с тактовой частотой 3.00 GHz и оперативной памятью 512Mb.

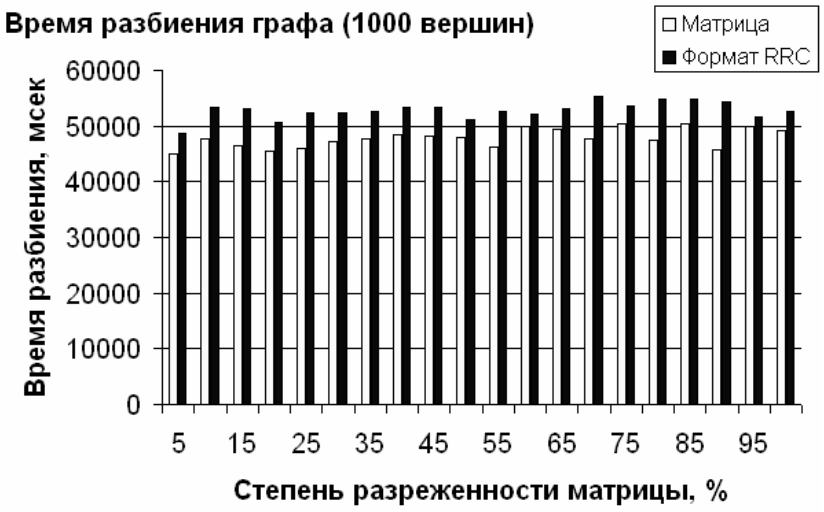


Рисунок 7. Разбиение графа, 1000 вершин

По графику видно, что разбиение графа с использованием формата RRCU занимает больше времени, чем с использованием двумерного массива. Однако, эта разница не существенна. Это обусловлено тем, что при использовании формата RR(C) усложняется доступ к элементу матрицы. Добавление ненулевого элемента к матрице в формате RR(C) или, наоборот, превращение элемента матрицы в 0 ведет за собой манипуляции с динамической памятью, что так же приводит к временным затратам.

Что касается экономии памяти, то использование формата RR(C) для разреженных матриц выгоднее чем использование двумерного массива. К тому же метод Хаусхолдера [7], использовавшийся при решении, подразумевает, что каждом шаге вычислений используется подматрица меньшей размерности, следовательно остальные элементы матрицы можно принять за 0 и не учитывать. Следовательно, на каждом шаге алгоритма объем занимаемой памяти будет уменьшаться. На рисунке 8 показано изменение занимаемой памяти для матрицы Лапласа в формате RR(C), размерностью 1000×1000.

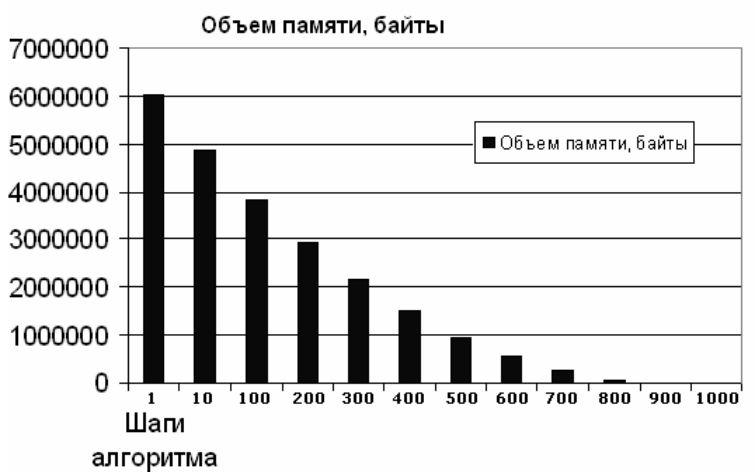


Рисунок 8. Объем памяти, занимаемый матрицей на различных шагах алгоритма Хаусхолдера

Список литературы

1. Bruce Hendrickson, Robert Leland. Multidimensional Spectral Load Balancing. Sandia National Laboratories Albuquerque, 1993
2. Bradford L. Chamberlain. Graph Partitioning Algorithms for DistributingWorkloads of Parallel Computations, 1998
3. А. Чернобаев. Графы. AGraph: библиотека классов для работы с помеченными графами. http://www.caravan.ru/~alexch/AGraph
4. Д. И. Батищев, Н. В. Старостиным. Методические указания по проведению лабораторных работ «задачи декомпозиции графов» по курсу «Эволюционно-генетические алгоритмы решения оптимизационных задач» для студентов факультета ВМК специальности «Прикладная информатика». Нижегородский государственный университет, 2001
5. Kirk Schloegel, George Karypis, Vipin Kumar. Graph Partitioning for High Performance Scientific Simulations.University of Minnesota, Department of Computer Science, Minneapolis, 2000

6. В. Быстрицкий. Представление разреженных матриц. http://alglib.sources.ru/articles/zeromatr.php.
7. Т. Шуп. Решение инженерных задач на ЭВМ: Практическое руководство. Пер. с англ. – М.: Мир, 1982. – 238 с

Дата надходження до редакції 21.10.2006 р.