электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель ФГБОУ ВПО "МГТУ им. Н.Э. Баумана". Эл No. ФС77-51036. ISSN 2307-0595

Структура данных для представления графов на параллельных вычислительных системах и параллельные алгоритмы операций над графами

 # 11, ноябрь 2014

Головков А. А., профессор Иванова Г. С.

УДК: 004.051+519.168

Россия, МГТУ им. Н.Э. Баумана
gsivanova@gmail.com

Введение

Теория графов находит широкое применение при решении различных комбинаторно-оптимизационных задач, существенную часть которых составляют NP-полные задачи. Алгоритмы их решения имеют высокую вычислительную сложность, как следствие требуют значительного времени, которое зависит от следующих взаимосвязанных факторов:

- архитектуры и характеристик вычислительных средств, для которых разрабатывается программа;
- способа машинного представления исходных и результирующих графов и эффективности реализации алгоритмов, встроенных в среду программирования.

Существенное увеличение быстродействия может обеспечить применение параллельных вычислительных систем. Такие системы, в отличие от обычных компьютеров, имеют некоторые особенности, которые связаны с архитектурой системы, в частности с организацией памяти и работой с ней.

1. Представление графов в памяти

Для представления графов в программных системах могут быть использованы матрицы смежности или инцидентности. Однако реализации на основе матриц при больших размерностях графов в реальных задачах приводят к неэкономному использованию памяти, так как в большинстве реальных задач эти матрицы сильно разрежены [1].

Соответственно более эффективным является аналитическое представление графов множествами вершин X и ребер U и множествами множеств их отображений по отношениям смежности FX, FU и инцидентности GX, GU.

В [1] рассмотрены различные варианты структур, используемые для реализации представлений графов:

- матрица;
- массив статических векторов;
- массив динамических векторов;
- список векторов;
- список списков;
- массив n-связных списков;
- список n-связных списков.

Указанные структуры широко применяются при разработке алгоритмов операций над графами. Исследования возможностей реализации параллельных алгоритмов и анализ применимости существующих структур данных, привели к необходимости разработки структуры данных для представления графа, ориентированной на параллельные вычислительные системы, с целью решения различных проблем, связанных с параллельным программированием, например:

- планирование работы – выделение для каждого потока отдельных областей структуры представления графа, например, вершины или ребра;
- одновременное использование памяти – возникновение конфликтов по данным: чтение после записи, запись после чтения, запись после записи.

Структура должна быть единой, централизованной и минимально иерархичной: вершины, ребра, их идентификаторы и веса, инцидентность вершин и ребер должны быть представлены в виде общей структуры с явной связностью, прозрачным доступом и сведенным к минимуму дублированием данных.

Множества, определяющие отношения инцидентности и смежности, вершины и ребра в графе, могут быть представлены в виде контейнеров соответствующего типа. Для их реализации целесообразно использовать массивы. Доступ к элементам контейнера будет осуществляться по индексу из разных потоков. Массив позволит эффективно реализовать атомарные операции добавления и удаления элемента, но привнесет трудности, связанные с динамическим изменением размера массива. За счет начального резервирования памяти большого объема для массива эти трудности могут быть преодолены.

Рассмотрим структуру данных для представления гиперграфа с количеством вершин n, количеством ребер m. Каждая вершина и ребро имеют идентификатор id от 0 до n - 1 и от 0 до m - 1 соответственно. Пример такого графа для n = 4 и m = 2 представлен на рисунке 1.

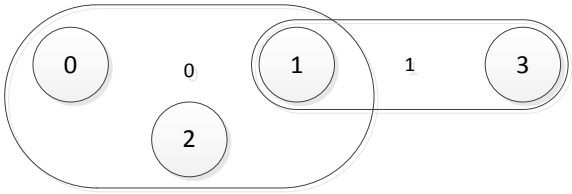

Рис. 1. Гиперграф

Гиперграф будет представлен структурой, показанной на рисунке 2. Она состоит из 2-ух указателей vertices и edges на массивы, содержащие указатели на вершины и ребра графа соответственно. Размеры этих массивов – verticesCount и edgesCount, равные n и m. Вершина или ребро графа описываются структурой, содержащей поля:

- id – идентификатор вершины или ребра;
- weight – вес вершины или ребра;
- connections – указатель на массив указателей инцидентных вершин или ребер;
- size – размер массива указателей инцидентных вершин или ребер.

За счет возможности реализации отношений вершин и ребер как многие ко многим и их двусторонней связности такое представление графа может быть расширено на реализацию любых графовых структур.

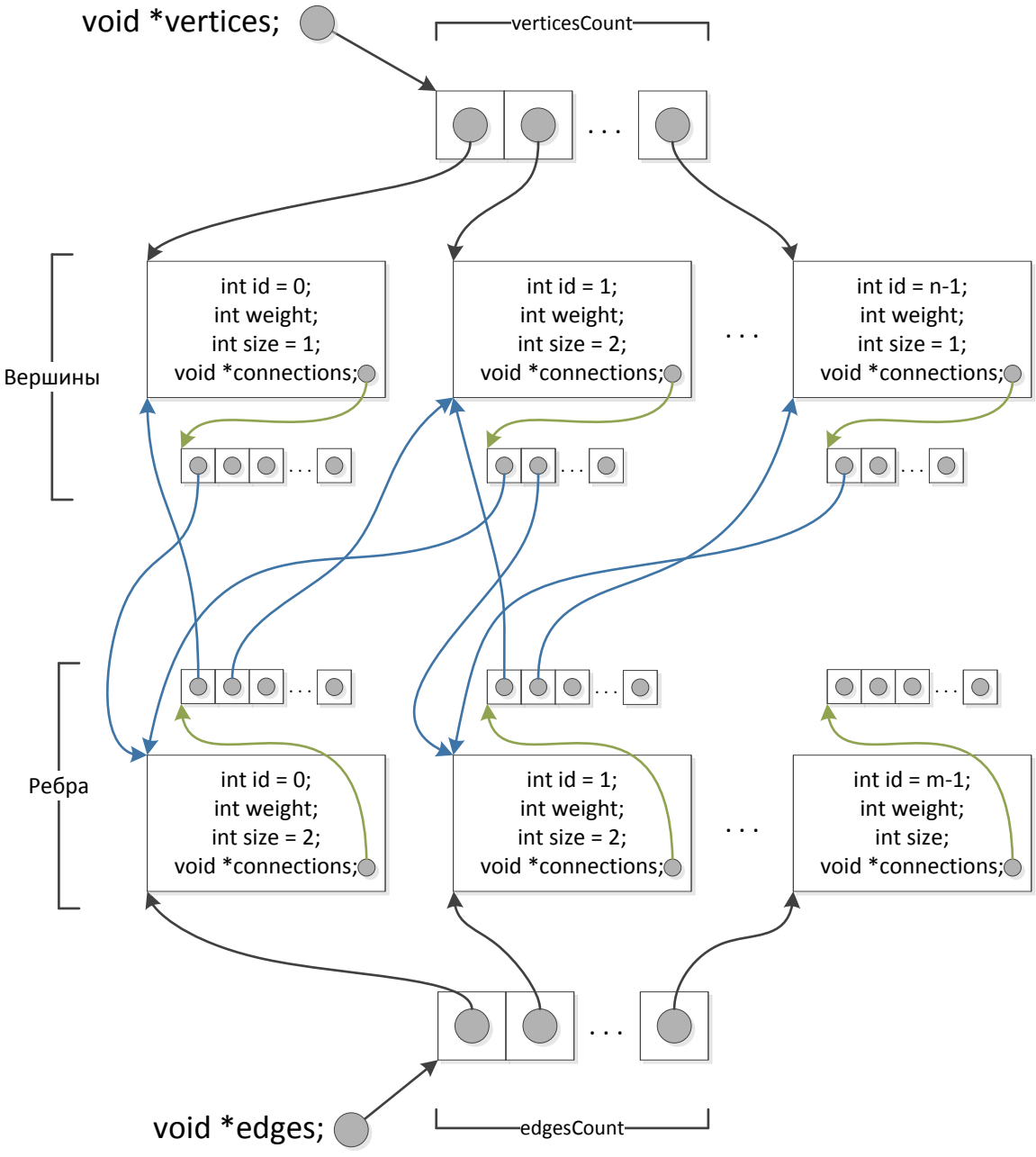

Рис. 2. Структура данных для представления графа

Данная структура имеет ряд преимуществ по сравнению с другими реализациями:

- ориентированность на параллельную реализацию алгоритмов работы со структурой — явно выделены структуры вершин и ребер, что дает возможность работать со структурой графа нескольким потокам, которые будут использовать свою отдельную вершину или ребро;
- высокая масштабируемость — представление вершины и ребра в виде структур позволяет при необходимости добавлять различные поля в структуру при описании алгоритмов;
- реализация массивами указателей на вершины и ребра, по сравнению с реализацией массивами самих вершин и ребер позволяет сократить время, необходимое для сдвига элементов массивов при удалении элемента из середины.

Как недостаток можно выделить отсутствие явного представления отношений смежности FX и FU, но они могут быть получены из отношений инцидентности GX и GU.

2. Параллельные операции над графами

Любой алгоритм решения задачи, которая требует применение теории графов, представляет собой совокупность вычислений характеристик графа и операций преобразования исходного графа в граф результата в соответствии с различными условиями. Операции над графами [2-4] обычно составляют большую часть тела разрабатываемого алгоритма.

Анализ операций показал наличие высокой степени параллелизма, в общем случае равной количеству вершин или ребер графа. Следовательно, для получения максимального ускорения обработки графовой модели необходимо использовать параллельную вычислительную систему с большим числом выполняемых параллельно задач. Таким требованиям подходят вычислительные системы с архитектурой CUDA [5].

Все потоки графического процессора CUDA могут работать только с памятью графического процессора, которая на сегодняшний день достигает нескольких гигабайт. Для того чтобы исключить копирование графа, определенного описанной выше структурой, из оперативной памяти в память GPU и обратно, она должна быть полностью расположена в памяти видеокарты.

Рассчитаем объем видеопамяти необходимый для хранения гиперграфа. Учитывая, что размер указателя равен 4 байтам, объем памяти для размещения всего графа в структуре, описанной выше, в байтах составляет:

$$M = 16 + 4(n + m) + 4(4 + size_n)\mathrm{n} + 4(4 + size_m)\mathrm{m}, \quad (1)$$

где 16 – суммарный размер указателей на массивы vertices и edges и переменных verticesCount и edgesCount;

n – количество вершин графа;

m – количество ребер графа;

$4(n + m)$ – размер массивов vertices и edges;

$size_n$ – средняя степень вершин графа;

$size_m$ – средняя степень ребер графа;

$4(4 + size_n)$n – размер структур вершин;

$4(4 + size_m)$m – размер структур ребер.

Полагая степени вершин $size_n$ и ребер $size_m$ равными 10 и 50 соответственно, (1) примет вид:

$$M = 16 + 60n + 220m. \quad (2)$$

График функции (2) представлен на рисунке 3. При объеме памяти графического процессора 4 ГБ предложенная структура позволит хранить графы с количеством вершин и ребер более 15 млн.

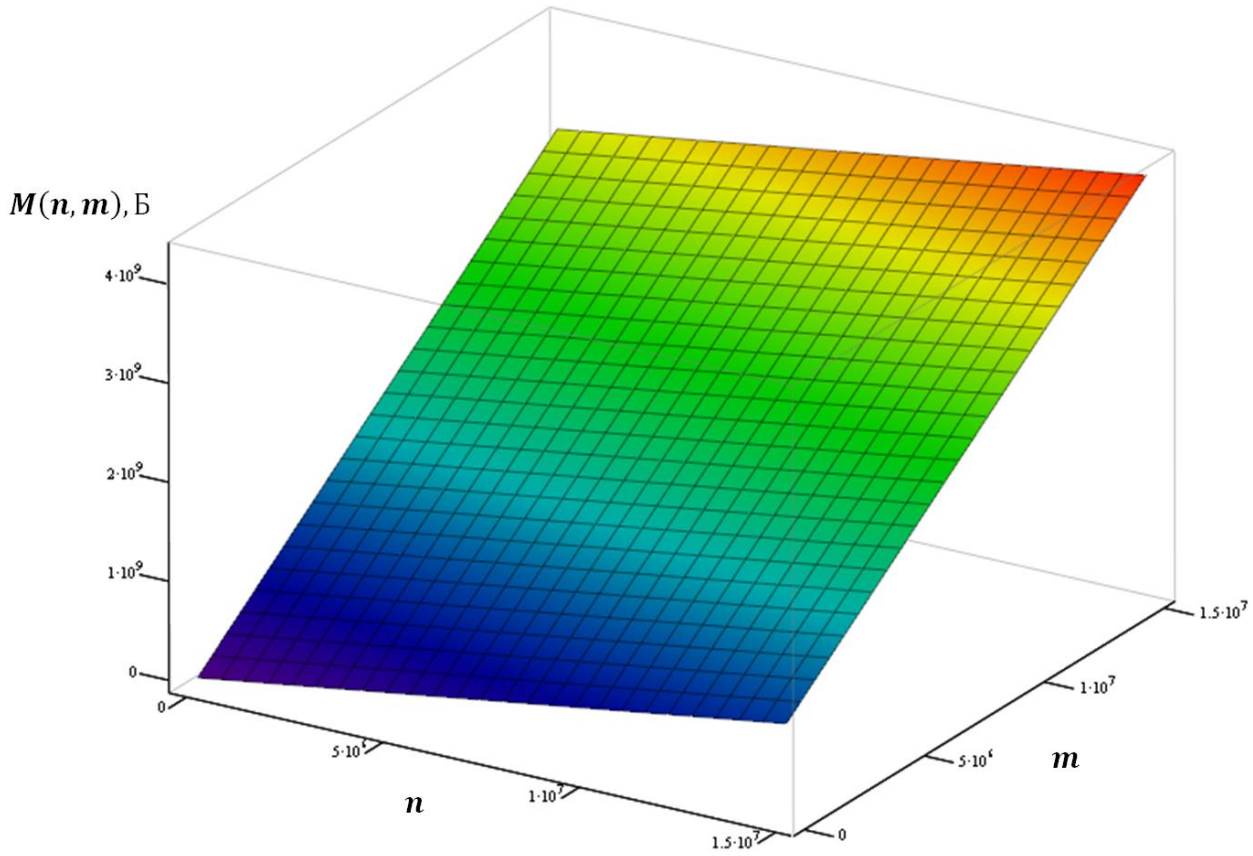

Рис. 3. График функции $M(n, m)$

Рассмотрим параллельный алгоритм операции добавления вершины в граф, так как эта простая с точки зрения реализации операция наиболее часто используется в различных алгоритмах. Сравнив быстродействие параллельного и последовательного варианта алгоритма такой операции можно получить более достоверные результаты об эффективности работы, как самого параллельного алгоритма, так и структуры, описывающей граф.

Последовательный вариант алгоритма подробно описан в [2]. В качестве исходных данных для выполнения операции имеем:

- X_k– вершина, которую необходимо добавить;
- $\Gamma_1 X_k$ – множество входящих ребер, которым инцидентна вершина Xk;
- $\Gamma_2 X_k$ – множество исходящих ребер, которым инцидентна вершина Xk.

Если не учитывать проверку применимости операции к графу и увеличение размеров массивов при необходимости, то алгоритм сводится к:

1) выделению памяти и инициализации вершины X_k, добавлению указателя на вершину X_k в массив vertices;
2) добавлению указателя на вершину X_k в массивы connections всех ребер из множества $\Gamma_1 X_k$;
3) добавлению указателей всех ребер из $\Gamma_2 X_k$ в массив connections вершины X_k.

При разработке параллельного алгоритма необходимо учитывать, что п. 1 необходимо выполнить до п. 2 и п. 3. При этом операции п. 2 и п. 3 целесообразно выполнить параллельно, разделив на 2 группы потоков, количество которых в группах будет равно размеру множеств $\Gamma_1 X_k$ и $\Gamma_2 X_k$ соответственно. Каждый поток из первой группы должен добавить указатель на вершину X_k в массив connections одного ребра из $\Gamma_1 X_k$, поток из 2 группы – указатель ребра из $\Gamma_2 X_k$ в массив connections вершины X_k.

Быстродействие такого параллельного алгоритма практически не зависит от количества вершин и ребер в исходном графе. Оно определяется суммой времени выполнения п. 1 и максимального времени работы одного потока из 1 или 2 группы.

3. Оценка быстродействия параллельного алгоритма

В процессе исследования был реализован параллельный алгоритм операции добавления вершины, на основе предложенной выше структуры, и последовательный алгоритм той же операции. В последнем случае граф описывался структурами типа массив массивов.

Для оценки времени выполнения операции последовательным и параллельным алгоритмом случайным образом генерировался гиперграф с различными значениями количества вершин n и ребер m, который для каждого значения n и m имел максимальную степень ребра, равную $\left[\frac{n}{4}\right]$, если $n > 8$, и 2 иначе. Ребра, инцидентные добавляемой вершине выбирались случайно, степень добавляемой вершины рассчитывалась как $\left[\frac{m}{4}\right]$, если $m > 4$, и 1 иначе.

Экспериментальные результаты времени выполнения параллельного алгоритма были получены на GPU Tesla C1060, последовательного – на CPU Intel Core i7-950. Графики времен выполнения приведены на рисунке 4: верхний график – последовательная операция, нижний – параллельная.

Время выполнения параллельной операции не зависит от n и m и в среднем составляет 0.024 мс. Время выполнения последовательной операции растет с увеличением n и m. На графиках видны пики, они соответствуют моментам прерывания центрального и графического процессоров на выполнение более приоритетных системных задач. График коэффициента ускорения представлен на рисунке 5. Сетка на графике соответствует единичному коэффициенту ускорения.

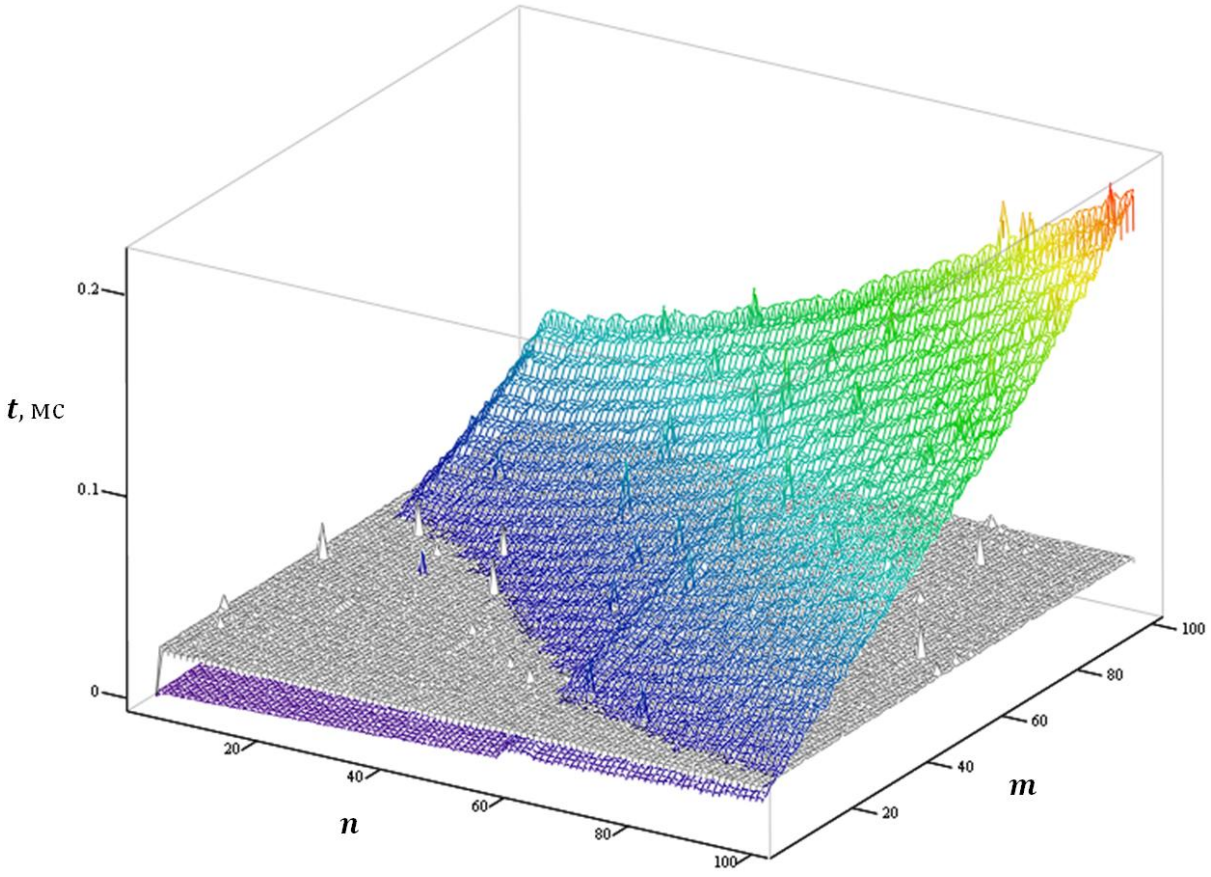

Рис. 4. Времена выполнения операции добавления вершины в граф

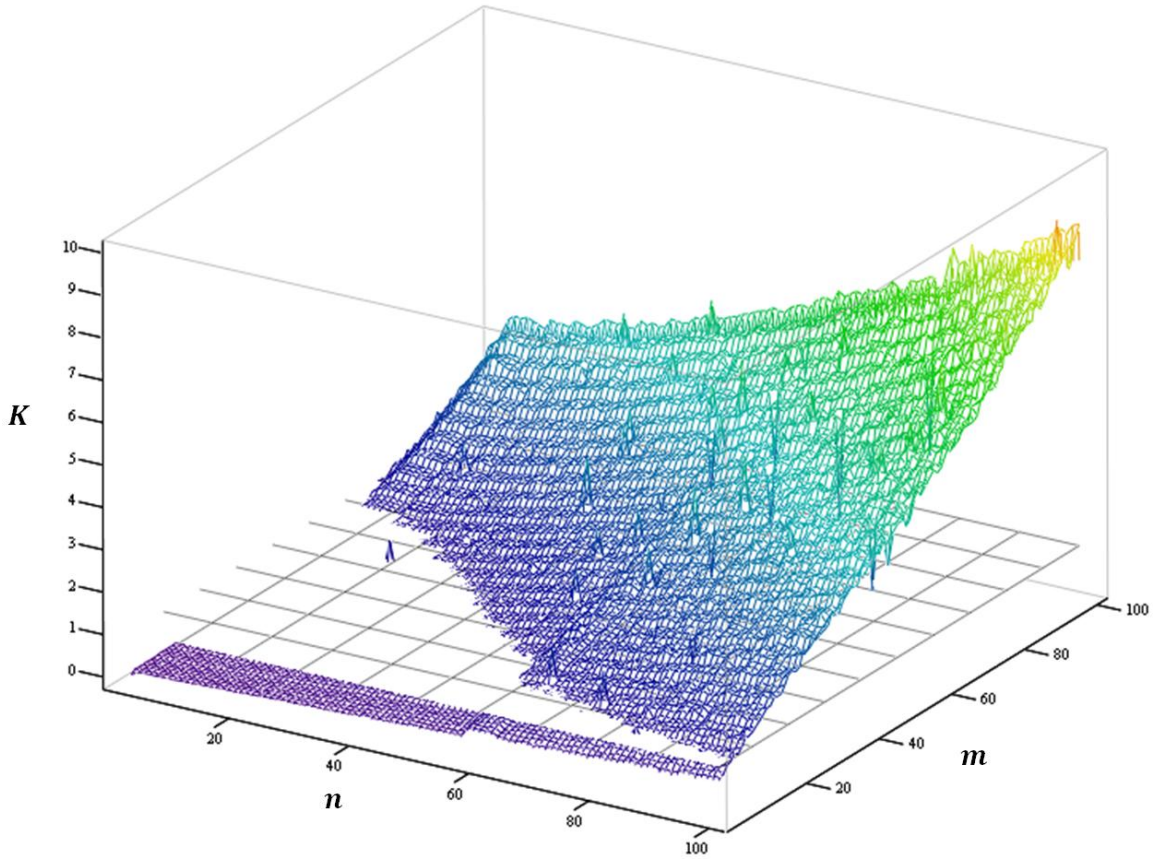

Рис. 5. Коэффициент ускорения

Заключение

Практические результаты показали, что предложенная структура представления графа полностью отвечает сформулированным требованиям к структуре, ориентированной на параллельные вычислительные системы, и может успешно применяться при реализации параллельных алгоритмов операций над графами в параллельных вычислительных системах. Уже при суммарном количестве вершин и ребер, равном 80, параллельный алгоритм добавления вершины выполняется быстрее, чем последовательный. С увеличением n и m значение коэффициента ускорения растет.

Применение параллельных алгоритмов на основе предложенной структуры описания графа открывает большие перспективы дальнейших исследований в направлении применения параллельных вычислений при решении задач над графами. Реализация параллельных операций над графами может существенно сократить время выполнения программ, обрабатывающих графы с большим количеством вершин и ребер и требующих значительных вычислительных мощностей.

Список литературы

1. Овчинников В.А., Иванова Г.С., Ничушкина Т.Н. Выбор структур данных для представления графов при решении комбинаторно-оптимизационных задач // Вестник Московского государственного технического университета им. Н.Э. Баумана. Приборостроение. 2001. № 2(43). С. 39-51.
2. Овчинников В.А. Операции над ультра и гиперграфами для реализации процедур анализа и синтеза структур сложных систем // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2009. № 10. Режим доступа: http://technomag.bmstu.ru/doc/132769.html (дата обращения 12.08.2014).
3. Овчинников В.А. Операции над ультра и гиперграфами для реализации процедур анализа и синтеза структур сложных систем (часть 2) // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2009. № 11. Режим доступа: http://technomag.bmstu.ru/doc/133223.html (дата обращения 12.08.2014).
4. Овчинников В.А. Операции над ультра и гиперграфами для реализации процедур анализа и синтеза структур сложных систем (часть 3) // Наука и образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2009. № 12. Режим доступа: http://technomag.bmstu.ru/doc/134335.html (дата обращения 12.08.2014).
5. Rob Farber. CUDA Application Design and Development. Waltham: Morgan Kaufmann, MA, USA. 2011. 336 p.