МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра компьютерных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Никулин Максим Игоревич

Разработка параллельных алгоритмов расчёта показателей надёжности сетей для исполнения на гибридных ВС

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Родионов А. С.

д. т. н, зав. лаб. ИВМиМГ СО РАН

…….
(подпись, дата)

Автор

Никулин М. И.

ФИТ, группа 9204

……..
(подпись, дата)

Новосибирск, 2013г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра компьютерных систем

УТВЕРЖДАЮ

Зав. кафедрой Пищик Б. Н.

…….
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Никулину Максиму Игоревичу

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема “Разработка параллельных алгоритмов расчёта показателей надёжности сетей для исполнения на гибридных ВС“

Цель работы: исследование вопросов связанных с разработкой точных и приближённых алгоритмов расчёта показателей надёжности сетей с ненадёжными элементами, предназначенных для параллельной реализации на гибридных кластерах, содержащих CPU и GPU.

Структурные части работы: изучение существующих последовательных алгоритмов расчёта показателей надёжности сетей с ненадёжными элементами, проведение экспериментов реализации параллельной версии алгоритма расчётов при помощи методов факторизации и полного перебора с использованием технологии CUDA.

Содержание

ВВЕДЕНИЕ

Случайные графы являются признанным инструментом моделирования ненадёжных сетей различного назначения. Существует много критериев надёжности сетей. Обычно выбор критерия надёжности осуществляется исходя из особенностей конкретной сети и её назначения. Связность сети, максимальная пропускная способность, время восстановления работоспособности и многие другие характеристики сети могут быть выбраны в качестве показателей надёжности. В большинстве случаев в качестве показателя надёжности моделируемых структур рассматривается вероятность связности выделенного подмножества вершин, далее называемая надёжностью графа. При равной вероятности присутствия ребра (надёжность ребра) имеем полиномиальное выражение надёжности графа через надёжность рёбра. Исследование этого полинома имеет как теоретический, так и практический интерес, поскольку позволяет выбирать более надёжные по рассматриваемому критерию структуры сетей путём сравнения графиков полиномов для различных структур. Получение такого полинома представляет собой NP-трудную задачу, поэтому исследуются различные приёмы ускорения расчётов.

Если элементы сети выходят из строя с некоторой вероятностью, то становится возможным оценивать математическое ожидание числа несвязных пар узлов сети (EDP, от англ. Expectation of the number of disconnected pairs of nodes). Далее критерий минимума математического ожидания числа несвязных пар вершин будем называть критерием минимума EDP или EDP-критерием.

Задача точного вычисления вероятности связности случайного графа с ненадёжными рёбрами представляла в большей степени чисто академический интерес, и на практике использовался приближенный метод. Однако развитие вычислительной техники привело к возрождению интереса к использованию точных методов на практике, так как появилась возможность за разумное время рассчитывать надёжность сетей малой и средней размерности (до десятков узлов).

Методы расчёта можно классифицировать следующим образом:

1. Приближенные методы:
   а) метод Монте-Карло.
   б) методы, основанные на рассмотрении остовов.
   в) методы, основанные на рассмотрении остовов.
2. Точные методы:
   а) методы факторизации, основанные на последовательном рассмотрении состояния рёбер.

б) методы, основанные на алгебре событий (комбинаторные)

в) методы последовательной редукции, основанные на эквивалентных преобразованиях графа (методы последовательно-параллельной редукции)

1 Математическая постановка задачи

G = (X, U) – неориентированный граф с множеством вершин X и множеством рёбер U.

n = |X| - число вершин графа.

m = |U| - число рёбер графа.

p_i - вероятность что ребро i исправно.

$q = (1 – p_i)$ – вероятность что ребро разорвано.

$R(p_1 \dots p_n)$ – вероятность того, что граф связен.

Случайный граф – это граф случайной структуры, имеющий случайное количество элементов (вершин и рёбер), если случайный граф взвешен, то веса его элементов также случайны.

Таким образом, распределение случайного графа G(n, m) определяется распределением вероятностей числа вершин (расположения рёбер) и весов элементов, т. е.

$P(G(X, U) = G^*(n, m)) = P(|X| = n)P(|U| = m)P(U(G) = U(G^*)) \times \Pi P(w_j(G) = w_j(G^*)) \Pi(g_{ij}P(G) = g_{ij}(G^*))$

Здесь w и g обозначают веса вершин и рёбер графа соответственно. При этом, пары вершин могут интерпретироваться как упорядоченные по направлению ребёр орграфа или неупорядоченные для обычного графа. Более простое определение случайного графа соответствует событию случайного выбора графа из некоторого множества графов.

Будем рассматривать неориентированные мультиграфы с равнонадёжными рёбрами, т. е. любое ребро графа G может быть удалено с вероятностью p. $0 <= p <= 1$.

Можно получить следующее аналитическое выражение для функции зависимости вероятности связности графа от значения p. Обычно этот полином представляют в виде $R(p) = p^m + \sum_{i=1}^{m-n+1} a_i \, (1-p)^i p^{m-i}$.

2 Технология CUDA.

Необходимо внести некоторые пояснения и описать основные особенности технология CUDA для нижеследующих алгоритмов. CUDA (с англ. Compute Unified Device Architecture) – программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia. В архитектуре CUDA используется модель памяти грид и кластерное моделирование потоков. Модель грид – это форма распределенных вычислений, в которой виртуальный суперкомпьютер представлен в виде кластеров, соединенных с помощью сети, слабосвязанных, гетерогенных компьютеров, работающих вместе для выполнения огромного количества заданий.

CUDA состоит из 2 типов процессоров, CPU и GPU. В роли CPU выступает обычный процессор компьютера, на нём выполняется главная часть программы, которая следит за выполнением подзадач и занимается управлением памятью. GPU процессоры физически расположены на видеокарте. В современных видеокартах содержатся несколько мультипроцессорных ядер, каждое из которых содержит большое количество процессорных ядер. Процессорные ядра никак не взаимодействуют с друг другом и не могут самостоятельно исполнять какие-то задачи. Все задачи на выполнение они получают от CPU. Процессы выполняемые на GPU ядрах в дальнейшем будем называть GPU-нитями или GPU-задачами. GPU процессоры обладают параллелизмом на уровне задач, поэтому видеокарту можно рассматривать как кластер компьютеров.

CUDA обладает несколькими типами памяти, мы же будем использовать напрямую только глобальную, так как исходные данные для всех GPU-нитей общие. Сама GPU-нить создает свои переменные в регистровой памяти, если же регистров не хватает, то переменные размещаются в разделяемой памяти. За этим следит сам GPU процессор, и мы никак не влияем на этот процесс. Упрощенная схема архитектуры CUDA приведена на рисунке 1.

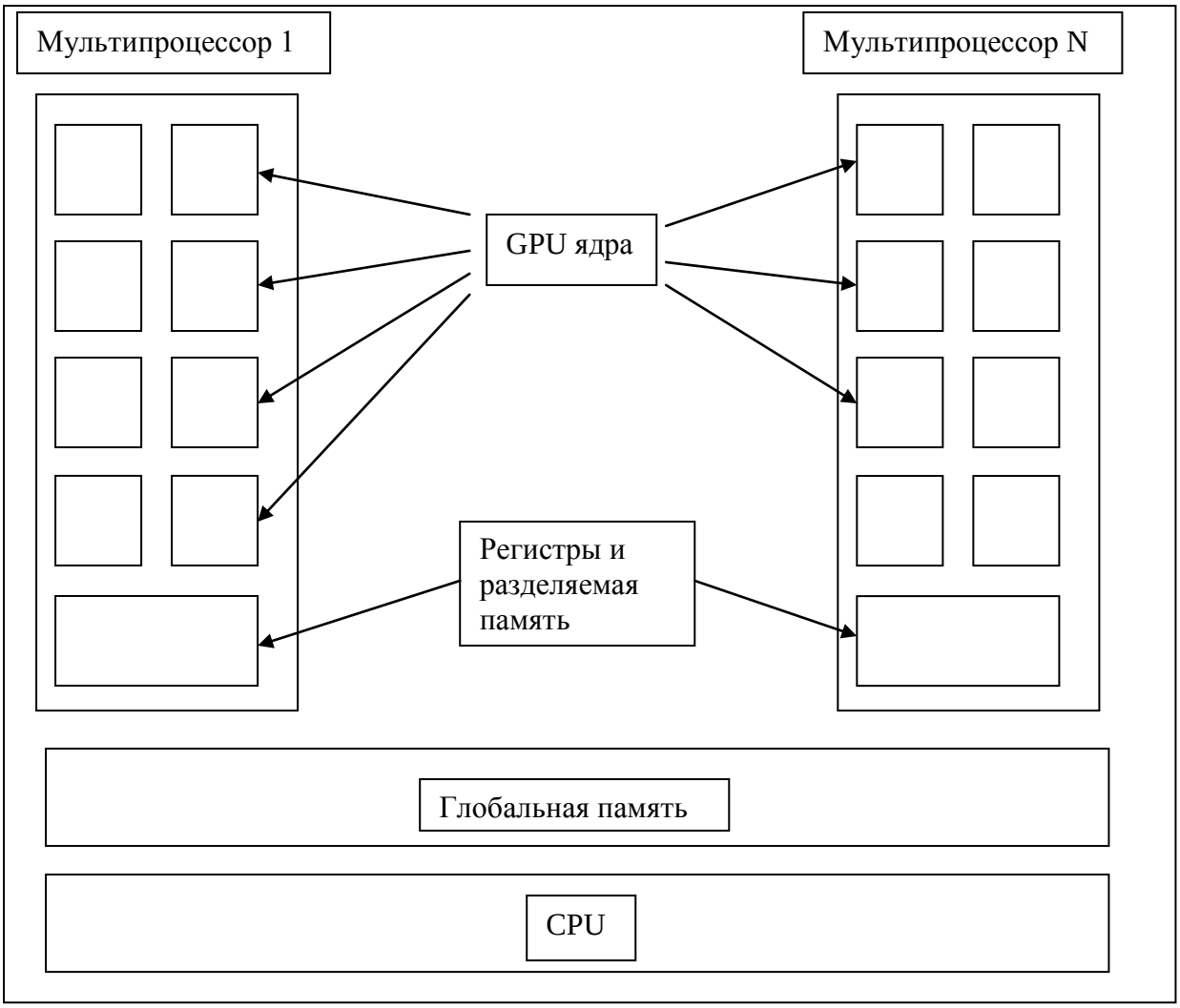

Рисунок 1. Упрощенная схема CUDA.

3 Метод факторизации.

Из точных способов определения вероятности связности графа с ненадёжными рёбрами наиболее широко известен метод факторизации или Мура-Шеннона. Ранее этот метод приводился в основном в качестве отрицательного примера, поскольку экспоненциальный рост числа рекурсий с ростом числа рёбер представлялся непреодолимым препятствием при рассмотрении графов реальной размерности. Из формулы полной вероятности следует следующее рекурсивное разложение графа. $R(G) = p_{ij}R\left(G^*(u_{ij})\right) + (1-p)R(G\ \{u_{ij}\})$, где $G^*(u_{ij})$ граф, стянутый по ребру u_{ij}, а $G\ \{u_{ij}\}$ граф, получаемый из G удалением ребра u_{ij}. Рекурсия продолжается до получения либо несвязного графа (возвращается 0), либо до получения графов малой размерности (2, 3 или 4 вершины), для которых вероятность легко вычисляется.

На рисунке 2 проиллюстрировано бинарное дерево разбиения графа G. Корнем дерево является граф G. Каждый переход по дереву означает удаление ребра i из графа с вероятностью p или стягивание с вероятностью (1 – p). Листьями дерева являются графы, для которых существуют простые формула расчёта связности либо несвязные графы. Последовательный подсчёт связности реализуется поиском в глубину по такому дереву.

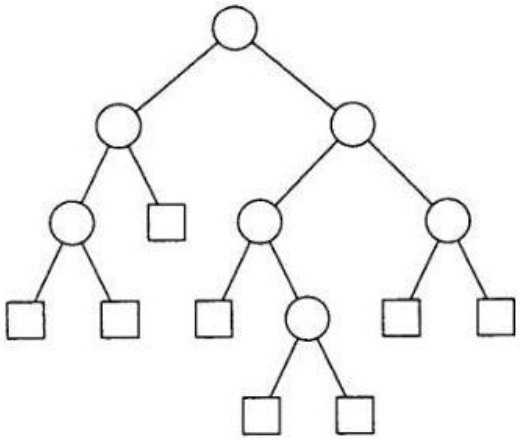

Рисунок 2. Бинарное дерево разбиений графа

Предлагалось рассмотреть вариант реализации метода ветвления с использованием покрывающего дерева. Идея алгоритма состоит в следующем: в графе строится произвольное покрывающее дерево T_0, после чего рассматриваются все возможные варианты его разрушения. Если удалить из него одно или несколько рёбер, то связность изначального графа G может быть обеспечена только за счёт остальных рёбер, все возможные комбинации которых и рассматриваются. При этом может появиться возможность снижения размерности за счёт учета параллельных рёбер, связывающих подграфы, образованные деревом. Основная формула получения вероятности связности

графа в рассматриваемом алгоритме имеет вид: R(G) = R(T_0) = $\sum_{i=1}^{N-1}\sum_{j=1}^{C_{N-1}^{i}} P_s\,(E_i^j)$, где $P_s\left(E_i^j\right)$ – вероятность того, что рассматриваемый граф связен при удалении из T_0i рёбер j-ым способом, умноженная на вероятность такого удаления рёбер. В событии E_i^j рассматриваются все возможные сочетания i рёбер.

3. 1 Пример

Рассмотрим работу алгоритма на графе с 4 вершинами и 5 рёбрами. Разбиение графа продемонстрировано на рисунке 3.

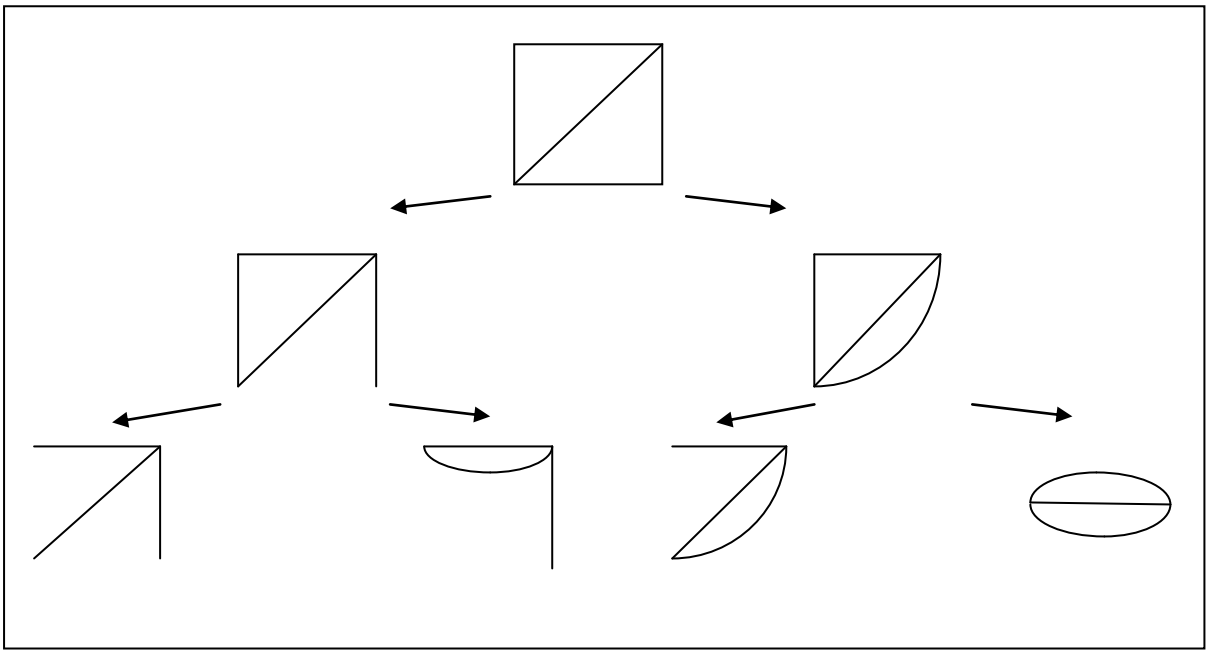

Рисунок 3. Пример применения метода факторизации.

При переходе влево мы с вероятностью p удаляем ребро из графа, при переходе вправо с вероятностью p-1 стягиваем вершины этого же ребра вместе. Данный процесс выполняется до тех пор, пока граф не распадётся на несвязные компоненты, либо не получится простой граф, для которого легко рассчитать вероятность связности.

3.2 Попытка реализации

Была предпринята попытка реализовать вышеуказанный алгоритм при помощи технологии CUDA. Программа, написанная с помощью CUDA, состоит из 2 частей: управляющий CPU-процесс и большое количество GPU-нитей. Связь поддерживается только между CPU и GPU, GPU-нити не могут напрямую сообщаться между собой и вызывать исполнение друг друга. По этой причине невозможно использование рекурсивных функций на GPU, а также приходится отправлять новые задачи по расчёту

через CPU (GPU -> CPU -> GPU). В виду большого времени, которое тратится на запись и чтение памяти между CPU и GPU, всё преимущество параллельного вычисления теряется. Большую часть времени исполнения программы занимают транспортные расходы. А также чтобы избежать холостого простоя GPU-нитей в ожидании задачи, требуется сложная и нетривиальная реализация синхронизации через общую память для GPU-нитей. Все эти причины привели к следующим выводам:

1. Реализовать расчёт вероятности связности графа метод факторизации на CUDA можно, но преимущества по времени расчёта не будет из-за транспортных расходов по сравнению с таким же алгоритмом, реализованным на MPI.
2. Реализация потребует сложных и нетривиальных знаний и способностей в программировании.

4 Метод полного перебора.

Поскольку преимущество CUDA состоит в быстром исполнении большого количества одинаковых задач, было решено попробовать реализовать расчёт вероятности связности методом полного перебора. За основу была взята следующая формула

$N(G) = \sum_{i=0}^{m} p^i (1-p)^{m-i} \sum_{j=1}^{C_m^i} N^*(G_{ij})$ (формула 3)

Где G_{ij} – j-ый вариант удаления из графа G ровно i рёбер (осуществляется с вероятностью $p^i(1-p)^{m-i}$), $N^*(G_{ij})$ – число несвязных пар вершин в G_{ij}. По сути, эта формула представляет собой подсчет математического ожидания в дискретном случае по определению, как сумма значений случайной величины, помноженных на вероятности случайной величины.

Алгоритм состоит из 2 частей, исполняемых на CPU и GPU соответственно. Задачей для каждой GPU-нити является построение разрушения исходного графа и вычисление $N^*(G_{ij})$ на полученном графе. CPU отвечает за генерацию управление памятью и запуск задач на GPU.

4. 1 Алгоритм на GPU.

Рассмотрим алгоритм расчётов на GPU более подробно. Исходными данными для каждой отдельной задачи являются:

1. Указатель на матрицу смежности исходного графа в памяти GPU - A.
2. Указатель на массив частичных сумм в памяти GPU - P.
3. Количество вершин в исходном графе - n.
4. Количество ребёр, которые нужно разрушить для построения разрушенного графа - i.

Необходимость передачи количества вершин n в дополнении к самой матрице смежности размера n^2 обуславливается особенностями языка реализации, так как только по указателю невозможно определить размер передаваемой матрицы. Вычисление задачи на GPU состоит из следующих этапов:

1. Вычисление порядкового номера GPU-нити - j. Номер вычисляется из координат нити в CUDA-блоке и координат блока в CUDA-решетке. Полученный номер однозначно определяется GPU-нитью и является уникальным. Num от 1 до C_m^i.
2. Копирование матрицы смежности для того чтобы не изменять исходную матрицу, общую для всех GPU-нитей.
3. Преобразование полученной целочисленной матрицы смежности в булевый тип, так как для него возможен более быстрый расчёт с использованием битовой маски.

B[k, l] = true, если A[k, l] > 0, иначе false. Где A – целочисленная локальная для данной GPU-нити матрица смежности. B – булева матрица смежности. k, l от 1 до n. n – количество вершин. Параллельно подсчитывается количество рёбер в графе – m.

4. Генерация сочетания i элементов из m, основываясь на порядковом номере num при помощи алгоритма генерации сочетаний в лексикографическом порядке. Алгоритм был выбран из соображений однозначного сопоставления порядкового номера GPU-нити сочетанию. В результате получаем уникальную выборку в каждой GPU-нити. Сложность алгоритма $O(b^2)$.
5. Удаление из булевой матрицы смежности рёбер из генерированной выборки.
6. Получение матрицы достижимости C из булевой матрицы смежности разрушенного графа при помощи алгоритма Флойда-Уоршелла. Сложность алгоритма $O(n^3/32)$. 32 – размер используемой битовой маски для ускорения вычислений.
7. Подсчёт количества несвязных пар вершин $N^*(G_{ij})$ в полученной матрице достижимости разрушенного графа. Сложность подсчёта $O(n^2/2)$.
8. Запись полученного значения $N^*(G_{ij})$ в массив частичных сумм по индексу, соответствующему порядковому номеру GPU-нити.

Результирующими данными каждой задачи, исполненной на GPU, является значение $N^*(G_{ij})$.

4.2 Алгоритм на CPU.

Алгоритм на CPU выполняется последовательно и запускается первым. Он является основой всей программы. Входными данными алгоритма являются количество вершин исходного графа и тип графа (полный, случайный, решетка) Вычисления на CPU состоят из следующих этапов:

1. Генерация целочисленной матрицы смежности A в памяти CPU. A[i, j] количество рёбер между вершинами i и j.
2. Копирование матрицы смежности в память GPU.
3. Расчёт оптимальных параметров CUDA-блоков и решеток.
4. Запуск задач на GPU-нитях с передачей им необходимых параметров.
5. После завершения исполнения всех нитей происходит копирование массивов частичных сумм на CPU.
6. Вычисление итогового результата с необходимыми коэффициентами (формула 3).
7. Очищение выделенной памяти.

Результатом работы алгоритма является точное значение EDP.

4.3 Структуры данных.

В реализации алгоритмы использовались следующие структуры данных:

1. Матрицы смежности представляют собой одномерный массив длиной n^2, где n – количество вершин. Одномерность массива используется для более быстрого последовательного доступа к элементам. Элемент A[i, j] преобразуется в A[i *n + j]. Используются матрицы смежности 2 типов: целочисленные и булевы. В целочисленном случае элемент A[i, j] хранит количество рёбер, соединяющих вершины i и j. В булевом случае элемент A[i, j] = true, когда существует хотя бы одно ребро, соединяющее i и j, иначе false. Так как рассматриваются ориентированные графы, то матрица смежности симметрична относительно главной оси. Матрицы формируются как в CPU так и в GPU.
2. Матрица достижимости представляет собой одномерный массив длиной n^2, где n – количество вершин. Используется матрица булева типа. A[i, j] = true, если существует хотя бы 1 путь из вершины i в вершину j. Матрица симметрична относительно главной оси аналогично матрицам смежности. Матрицы используется только в процессах, исполняемых на GPU.
3. Массив частичных сумм. Используются массивы частичных сумм различной длины. При переборе разного количества удаляемых рёбер из исходного графа формируются массивы длиной C_m^i. Где i – количество удаляемых рёбер, m – общее количество рёбер. Массив имеет целый тип. Память для них выделяется и очищается динамически внутри основного цикла на CPU.
4. Сочетания удаляемых рёбер представляют собой целочисленный массив, содержащий номера рёбер, которые удаляются в конкретной GPU-нити. В силу генерирующего алгоритма, рёбра отсортированы в порядке возрастания. Эти массивы используются только в GPU.

4.4 Среда разработки и окружение.

Для разработки использовалась следующее программное обеспечение и оборудование:

1. Microsoft Visual Studio 2010 C++ - стандартная среда разработки на языке C/C++.
2. Nvidia Nsight Visual Studio Edition 3.0 – отладчик и анализатор кода.
3. CUDA SDK 5.0.
4. CPU Intel Core i5-3210M 2.50 GHz.
5. GPU Nvidia GeForce GT 530m.

а) 2 мультипроцессора

б) частота ядер 700 MHz.

в) Частота процессора 1.4 MHz.

г) 96 * 2 процессорных ядер.

д) 1Gb видеопамяти

е) Производительность 268.8 GFLOPS

Алгоритм реализован на специальном упрощенном диалекте языка программирования C, выполнимом на графических процессорах Nvidia.

4.5 Численные результаты.

Была проведена серия расчётов EDP для графов разной размерности и построен следующий график зависимости времени расчётов от числа рёбер в случайном графе (рисунок 4). Данный график был построен на графах с 15 вершинами. Однако при другом количестве вершин (8, 10, 12) время исполнения изменяется незначительно по сравнению с изменением количества рёбер. Разница составляет 10-15% от общего времени выполнения расчётов. От количества вершин зависит время выполнения на GPU-нити, а от количества рёбер – количество запускаемых GPU-нитей.

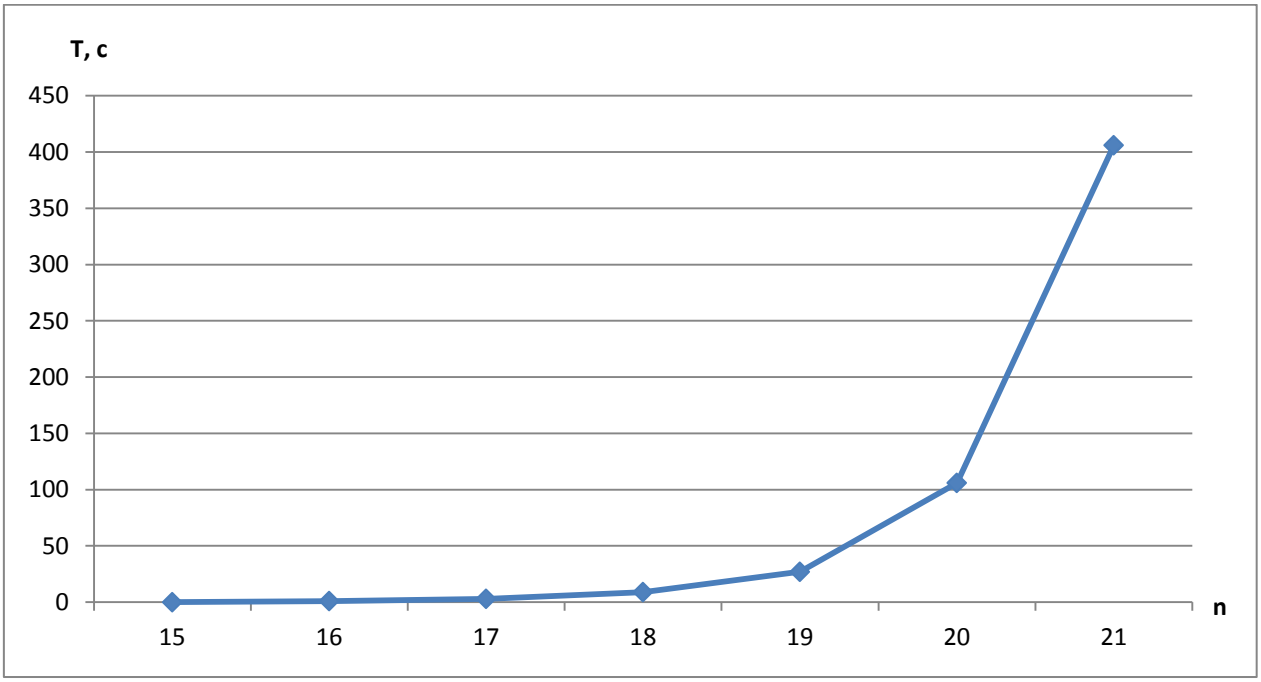

Рисунок 4. Зависимость времени расчётов от количества рёбер графа

На графике четко прослеживается экспоненциальная зависимость. Отдельные GPU-нити исполняются очень быстро, однако с ростом количества вершин, экспоненциально растёт количество вариантов удаления рёбер из графа. Физических мощностей графических видеокарты не хватает, чтобы запустить одновременно все задачи, поэтому задачи вынуждены ожидать освобождения GPU-нитей. Отсюда и получаем замедление

расчётов. Можно воспользоваться более мощными видеокартами с большим количеством одновременных потоков, чтобы ускорить вычисления, однако предел есть и у них и при каком-то количестве вершин в графе расчёт будет нецелесообразным при методе полного перебора.

ЗАКЛЮЧЕНИЕ

Целью данной работы было исследование возможностей разработки точных и приближённых алгоритмов расчёта показателей надёжности сетей с ненадёжными элементами и проведение численных экспериментов на гибридных вычислительных системах. Поставленные цели были достигнуты: в работе описаны причины сложности реализации и нецелесообразность использования метода факторизации, а также возможность расчёта показателей надёжности графов средней размерности используя метод полного перебора совместно с технологией CUDA. Размерность графа, для которого возможно произвести расчёт за приемлемое время, зависит от мощности используемой видеокарты, в большей степени от количества одновременно исполняемых потоков.

Список литературы

1. Родионов А.С. К вопросу ускорения расчёта коэффициентов полинома надёжности случайного графа. Автоматика и телемеханика. №7. 2011. С. 134–146.
2. Родионова О.К. Исследование и разработка методов анализа и синтеза оптимально-связных информационных сетей. Новосибирск, ИВМиМГ СО РАН, 2003.
3. Мигов Д.А. Расчёт надёжности сетей. Новосибирск, ИВМиМГ СО РАН, 2008.
4. Липский В.И. Комбинаторика для программистов. Москва, Мир, 1988.
5. Документация по CUDA http://docs.nvidia.com/cuda.
6. Онлайн система вопросов и ответов http://stackoverflow.com.