Московский государственный университет имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра суперкомпьютеров и квантовой информатики

Козлов Михаил Владимирович

**ТЕМА**

Научный руководитель:

доцент, к.ф.-м.н.

Попова Нина Николаевна

Москва, 2017

Это черновой вариант курсовой работы, в нём нет картинок и не указаны источники. Некоторых параграфов, возможно, нет, а некоторые без текста, так как ещё не понятно, что там писать.

Так же прямо в тексте есть комментарии от меня, *(КОТОРЫЕ ВЫГЛЯДЯТ ВОТ ТАК)*. Некоторый текст я буду немного переписывать (добавлять\удалять), так как он может быть просто скопирован с других статей. Форматирование текста в данном файле плохое. Формулы набраны просто текстом.

Прошу Вас написать прямо в этом файле комментарии по поводу параграфов, которые могут быть написаны не корректно или плохо.

**ВВЕДЕНИЕ**

Современные вычислительные задачи науки и техники являются настолько трудоёмкими, что уже не могут обходиться без параллельных вычислений. Для решения таких задач необходимо иметь огромное количество вычислительных мощностей и памяти, что привело к созданию многопроцессорных вычислительных систем. Такие системы представляют из себя вычислительные узлы, соединенные локальной коммуникационной сетью. Так как в вычислительной задаче не всегда возможно обеспечить локальность всех необходимых данных, узлы системы могут взаимодействовать друг с другом через коммуникационную сеть, передавая сообщения с нужными данными. Производительность суперкомпьютеров (измеряется количеством выполняемых операций с плавающей точкой в секунду (floatingpoint operations per second, FLOPS, ФЛОПС) достигается как увеличением производительности вычислительных узлов, так и увеличением количества узлов и количества вычислительных ядер внутри узла.

В настоящее время вычислительные системы из первой десятки списка TOP500 суперкомпьютеров включают от сотен тысяч до миллионов ядер и их производительность составляет десятки петафлопс. К 2020-ым годам планируется создание первых экзафлопсных суперкомпьютеров, которые предположительно будут иметь от сотен тысяч до миллионов вычислительных узлов, а каждый узел будет иметь от сотен до тысяч ядер (как следствие суперкомпьютеры будущего будут иметь еще более сложные коммуникационные сети). Но при этом количество прямых связей между процессорами остаётся относительно постоянным, так как при росте количества процессоров очень сложно обеспечить полносвязность системы: число связей контактов каждого процессора возрастает линейно, а общее число связей в системе – квадратично. Время выполнения параллельных приложений в топ числе зависит от времени обмена данными между процессами. При росте количества процессоров системы увеличивается и максимальные расстояние между любой парой процессоров (диаметр системы), что может привести к увеличению времени пересылки сообщения, что, в свою очередь, приводит к снижению эффективность работы суперкомпьютера. Если частовзимодействующие процессы расположить ближе друг к другу, это снизит время коммуникации между ними, и в следствие уменьшит время выполнение программы. Оптимальное расположение процессов параллельной программы по вычислительным узлам системы – интуитивное определение задачи мэппинга, о которой пойдёт речь в следующем параграфе.

**ЗАДАЧА МЭППИНГА**

Формально задачу мэппинга можно описать следующим образом: Пусть *G =* (*V G* , *E G* ) граф параллельного приложения. Вершины *V G* = {1, 2, …, N} параллельные процессы программы. Ребра *E G* показывают наличие или отсутствие коммуникационных взаимодействий между парами процессов. Каждое ребро (*i* , *j* ) имеет вес *w* (*i* ,*j* ), который показывает интенсивность коммуникационных взаимодействий между процессами *i* и *j*. В простейшем случае необязательно вводить вес ребер, можно обойтись просто их наличием/отсутствием. В качестве интенсивности коммуникационных взаимодействий могут выступать различные величины, например: суммарное количество передаваемых данных между процессами, суммарное количество вызовов функций передачи сообщений за все время выполнения параллельного приложения или частота вызовов функций передачи сообщений за определенный промежуток времени. Далее вводится граф вычислительной системы *H* ( *V H* , *E H* ), в котором *V H* множество

вершин, соответствующее вычислительным узлам, а ребра *E H* каналы

коммуникационного взаимодействия между вычислительными узлами.

Мэппингом назовем отображение вершин графа параллельного приложения *G* на вершины графа вычислительной системы *H*

*T* : *V G* ⟶ *V H* .

**АКТУАЛЬНОСТЬ МЭППИНГА**

Основное преимущество мэппинга состоит в том, что с его помощью можно добиться увеличения производительности отдельно взятой параллельной программы, не внося изменений в исходный код. С появлением экзафлопсных суперкомпьютеров проблема мэппинга может стать особенно актуальной, так как ожидается значительное увеличение количества вычислительных узлов.

Так же задача мэппинга может снизить энергопотребление суперкомпьютера.

**СУЩЕСТВУЮЩИЕ ПОДХОДЫ К РЕШЕНИЮ ЗАДАЧИ МЭППИНГА**

Поиск оптимального размещения (мэппинга), вообще говоря, является NP-сложной задачей. Существуют различные подходы при создании точных алгоритмов, но все они имеют экспоненциальную сложность, из-за чего применимы только для небольших программ и небольших коммуникационных сетей. Некоторые из точных алгоритмов: Branch and Bound (Метод ветвей и границ), Breadth first search (Поиск в ширину), Best first search (Поиск «первый – лучший»), A\* (А-звездочка).

Один из самых простых и часто встречающихся алгоритмом является жадный алгоритм. Они имеют полиномиальную сложность, но находят локально оптимальное решение. Простейший пример жадного алгоритма для решения задачи мэппинга: Случайным образом выбирается процесс параллельного приложения и вычислительный узел. Выбранный процесс размещается на вычислительном узле. Далее выбирается процесс сосед с которым есть взаимодействие. Этот процесс размещается на ближайшем вычислительном узле.

Еще один часто встречающийся способ решения задачи мэппинга разбиение

графа. Граф параллельного приложения и граф коммуникационной сети рекурсивно делятся на два подграфа. Далее на выходе из рекурсии процессы размещаются на вычислительных узлах.

Итерационные алгоритмы направлены на поиск оптимального решения посредством улучшения начального решения. Начальное решение, вообще говоря, может быть получено с использованием жадных алгоритмов. Обычно итерационные алгоритмы меняют расположение процессов, чтобы получить локально лучшее решение. Во многих итерационных алгоритмах используются случайные возмущения решений, чтобы таким образом избежать локально оптимальное решение.

Эволюционные алгоритмы – алгоритмы для решения задач оптимизации, в частности задачи мэппинга, основанные на принципах природной эволюции различных процессов. К эволюционным алгоритмам относятся генетический алгоритм, роевые алгоритмы, алгоритмы с использованием нейронных сетей и другие.

**АРХИТЕКТУРЫ КОММУНИКАЦИОННЫХ СЕТЕЙ**

В данном разделе рассмотрим наиболее популярные топологии коммуникационных сетей, ссылки на которые часто используются при обсуждении задачи мэппинга. Рассмотрим некоторые, наиболее популярные топологии, используемые в суперкомпьютерах, входящих в список TOP500. Оценку сетей будем проводить, определяя основные параметры: диаметр сети, определяемый как максимальное расстояние между любыми двумя узлами сети, и пропускная способность бисекции (bisection bandwidth) – минимальная общая пропускная способность линий связи, которые придется разрезать для того, чтобы разделить сеть на две равные части. С ростом числа узлов диаметр сети растет. Для многих топологий бисекционная пропускная способность уменьшается относительно общего числа узлов.

**ЖИРНОЕ ДЕРЕВО**

Архитектура сети «жирное дерево» имеет структуру дерева с обрабатывающими элементами, расположенными в листьях и переключательными элементами в промежуточных узлах. С продвижением по дереву от узлов вверх пропускная способность линий связи, реализующих ребра дерева, увеличивается, делая их «жирнее».

Реализация архитектур жирное дерево поддерживается коммуникационными сетями Infiniband, интерконнектом Federation (IBM). На практике обычно используется несколько переключателей на верхнем уровне.

**РЕШЕТКА И ТОР**

В сетях с топологией n-размерных решеток каждый процессор связан с двумя другими процессорами в каждом измерении. Это дает общее число 2n связей у каждого процессора. Наиболее часто используемый вариант решеток – это решетки с n=3, или 3D решетки. Для 3D решеток с размером N в каждом измерении диаметр решетки равен 3N. На n-мерных решетках реализуются топологии типа тор путем циклического замыкания связи процессоров, находящихся на концах строк/столбцов процессорной решетки.

Примером реализации n-мерных решеток является коммуникационная подсистема обмена 2-ухсторонними сообщениями в архитектуре Blue Gene/P [ 4, Архитектура BGP]

**ПРЯМЫЕ ИЕРАРХИЧЕСКИЕ СЕТИ**

Прямые иерархические сети являются примером новых современных типов коммуникационных сетей, отличающихся гораздо большей bisections bandwidth по сравнению с традиционно используемыми топологиями 3-ехмерных торов. Топология этих сетей может быть представлена как декартово произведение так называемых all-to-all сетей. Существуют две реализации этих сетей. IBM определяет их как PERSC (Productive Easy-to-use Rekiabke Computing System), используя это имя как название системы, так и как топологии сети [5]. Система Cascade разработки компании Cray является другим примером прямых иерархических сетей. Топология этой сети известна как dragonfly топология [6].

Основное отличие Cascade и PERSC сетей заключается в том, что PERSC является 2-уровневой прямой сетью, а dragonfly – 3-уровневой. Dragonfly топологии являются примерами сетей с высокой степенью масштабируемости и хорошим соотношением цены и качества.

**IBM BLUE GENE/P**

Все тесты моего алгоритма проводились на суперкомпьютере IBM BLUE GENE/P, поэтому здесь приводится его описание с сайта *(ССЫЛКА НА САЙТ)*.

С 2008 года на факультете ВМК МГУ имени М. В. Ломоносова работает суперкомпьютер IBM Blue Gene/P, который является одной из первых систем данной серии среди установленных в мире. Архитектура Blue Gene была предложена компанией IBM в рамках проекта по исследованию возможностей достижения новых рубежей в супервычислениях. Более крупные машины данной серии в настоящее время занимают лидирующие позиции в списке пятисот самых мощных компьютеров мира Top500, а машина Blue Gene/P, установленная на ВМК МГУ, в редакции рейтинга от 18 ноября 2008 года оказалась на 128-м месте (в редакции от 16 ноября 2009 года — 348-е место). В списке самых высокопроизводительных компьютеров стран СНГ, опубликованном 22 сентября 2009 года, она находится на 4-й строчке.

Система IBM Blue Gene/P принадлежит к новому семейству суперкомпьютеров, обладающих высокой производительностью, масштабируемостью, возможностью обрабатывать данные большего объема, потребляя при этом значительно меньше энергии и занимая меньшую площадь по сравнению с предыдущими системами.

На факультете ВМК МГУ представлена конфигурация, состоящая из двух стоек, содержащих в общей сложности 2048 вычислительных узлов, каждый из которых включает четыре ядра PowerPC 450, работающих на частоте 850 мегагерц, что дает пиковую производительность 27,9 терафлопс (триллионов операций с плавающей точкой в секунду).

Архитектура Blue Gene спроектирована для вычислительных кодов, которые хорошо масштабируются до сотен и тысяч процессоров. Несмотря на то, что индивидуальные процессорные ядра системы Blue Gene работают на относительно низкой частоте, для приложений, способных эффективно использовать большое число процессорных элементов, удается достигнуть значительно более высокой производительности по сравнению с традиционными суперкомпьютерами.

*(ВОЗМОЖНО НУЖНО БУДЕТ ДОБАВИТЬ И НЕМНОГО ПЕРЕПИСАТЬ)*

**ЦЕЛИ РАБОТЫ**

*(ЕЩЁ НЕ ЗНАЮ, ЧТО ПИСАТЬ, КРОМЕ РАЗРАБОТКИ АЛГОРИТМА МЭППИНГА)*

**ОПИСАНИЕ АЛГОРИТМА**

*(ПРИМЕРНОЕ ОПИСАНИЕ)*

На вход программе подаётся коммуникационная матрица программы для оптимизации. Каждая строка матрицы сортируется и создаётся мэппинг-вектор, вектор содержащий по 6 наиболее оптимальных соседей для данного процесса. Далее выбирается самый нагруженный (по пересылкам) процесс и расставляется на трёхмерный тор (реализован как куб (трёхмерный массив), с функцией поиска соседей, которая учитывает, что крайние процессы тоже соединены). После этого берется один из соседей и то же самое делается для него. В конце, на свободные мест, расставляются процессы, которые ни с кем не взаимодействуют (если такие есть). После этого из трехмерный массив переводиться в одномерный, сортируется и выводиться в файл, который и будет мэппингом параллельной программы.

Так же были написаны несколько программ для анализа «собственных трасс», по которым я строил коммуникационные матрицы программы, для алгоритма мэппинга. Создаётся копия программы, в которую заносится функция открытия файла с именем «proc\_MYRANK», в который будут выводиться данные вида: кому осуществляется отправка, объём. Пока инструментария программы проводится вручную *(ПОПРОБУЮ УСПЕТЬ НАПИСАТЬ ПРОГРАММУ ДЛЯ ЭТОГО)*. Далее по получившимся файлам строится коммуникационная матрица, а по ней мэппинг.

Для самой тестовой программы *(ХОЧУ ЕЁ НЕМНОГО УСЛОЖНИТЬ)*, создаётся рандомный вектор с номерами процессов, с которыми будет взаимодействие. После чего этот вектор сохраняется и на нём проводятся тесты (создание трассы, мэппинга, замер времени). Потом можно создать новый вектор.

**ТЕСТЫ И РЕЗУЛЬТАТЫ**

*(ПОКА ЧТО БЫЛА ПРОТЕСТИРОВАННА ТОЛЬКО ОДНА ПРОГРАММА, КОТОРУЮ Я НАПИСАЛ САМ В ВИДЕ ТЕСТА, БЕЗ КАКОЙ-ЛИБО ПОЛЕЗНОЙ НАГРУЗКИ)*

Было получено ускорение до 39% на тестовой программе, в зависимости от распределения процессов(рандомного) и объёма данных для пересылки.

Со случайным мэппингом тестирование ещё не проводилось.

**ПЛАНЫ НА ДАЛЬНЕЙШУЮ РАБОТУ**

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

<http://hpc.cmc.msu.ru/bgp>

<http://russianscdays.org/files/pdf16/95.pdf>

<http://www.mkurnosov.net/uploads/Main/mkurnosov-vestnik-sibsutis09.pdf>

<http://cpct.sibsutis.ru/~apaznikov/uploads/Main/paznikov-kurnosov-kupriyanov2015-problem-info.pdf>

*(БУДУТ ЕЩЁ)*