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

Кафедра параллельных вычислений

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Сафин Альберт Рустемович

Алгоритмы редукции для системы фрагментированного программирования

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель	**Автор**
Киреев С.В.	Сафин А.Р.
(фамилия , И., О.)	(фамилия , И., О.)
	ФИТ, 9204
(уч.степень, уч.звание)	(факультет, группа)
…………...	……………..
(подпись, дата)	(подпись, дата)

Новосибирск, 2013 г.

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

Кафедра параллельных вычислений

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, И., О.)

…………………
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Сафину Альберту Рустемовичу

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: Алгоритмы редукции для системы фрагментированного программирования.

Исходные данные (или цель работы): разработать алгоритм асинхронной распределенной редукции для использования в системе фрагментированного программирования LuNA.

Структурные части работы: Изучение технологии фрагментированного программирования. Обзор алгоритмов редукции. Описание модели редукции в системе исполнения фрагментированных программ. Разработка параметризованного алгоритма редукции. Реализация эмулятора работы алгоритма для оценки характеристик его работы при различных параметрах.

Оглавление

ВВЕДЕНИЕ **5**

1 Основные определения **7**

1.1 Технология фрагментированного программирования 7

1.2 Операция редукции 10

1.2.1 Формальная модель редукции 10

1.2.2 Модель редукции в ТФП 11

1.3 Постановка задачи 13

2 Обзор алгоритмов редукции **14**

2.1 Реализация редукции в MPI 14

2.1.1 Алгоритмы редукции в Open MPI 15

2.1.2 Алгоритмы редукции в MPICH 15

2.2 Реализация редукции в Charm++ 16

2.3 Выводы 17

3 Разработка алгоритмов параллельной асинхронной редукции **19**

3.1 Идеи алгоритма 19

3.2 Построение дерева вычислителя 20

3.3 Алгоритм работы узла 21

3.3.1 Алгоритм для полной редукции 22

3.3.2 Алгоритм для частичной редукции 23

4 Оценка алгоритмов **25**

4.1 Эмулятор работы алгоритмов редукции 25

4.2 Исследование параметров алгоритмов 26

4.2.1 Полная редукция 26

4.2.2 Частичная редукция 27

Заключение **29**

Литература **30**

ВВЕДЕНИЕ

Решение большинства современных задач науки и техники связаны с крупномасштабным численным моделированием. До сих пор многие параллельные программы для мультикомпьютеров создаются с использованием средств параллельного программирования низкого уровня. В результате разработчики каждый раз заново решают многие задачи системного параллельного программирования, делая параллельную реализацию трудоёмкой, зачастую неэффективной и подверженной многим ошибкам.

Причиной этого является отсутствие средств параллельного программирования достаточно простых, чтобы ими пользовались люди без знания тонкостей параллельного программирования, и достаточно эффективных, чтобы качественно отобразить сложный алгоритм на параллельную вычислительную систему. В связи с этим актуальной является задача развития технологии параллельного программирования, повышения уровня программирования и эффективности создаваемых параллельных программ. В идеале по высокоуровневому описанию алгоритма должна автоматически конструироваться эффективная параллельная программа.

В ИВМиМГ СО РАН в лаборатории Синтеза параллельных программ разрабатывается технология фрагментированного программирования, целью которой является достижение перечисленных выше целей. Она основывается на методе синтеза параллельных программ [1]. В рамках этой технологии разрабатывается система фрагментированного программирования LuNA [2]. Одной из задач при ее разработке является создание эффективных параллельных реализаций для высокоуровневых операций алгоритма.

Целью дипломной работы является разработка и реализация алгоритма асинхронной параллельной редукции (свёртки) для системы LuNA. В работе был выполнен обзор, показывающий, что реализация редукции для системы фрагментированного программирования должна отличаться от реализации редукции в существующих системах параллельного программирования. В результате работы был спроектирован алгоритм для операции редукции двух видов: полной (с единственным результатом) и частичной (с последовательностью ча-

стичных результатов). С помощью моделирования были определены оптимальные параметры алгоритма для различных характеристик вычислительной системы.

Глава 1

Основные определения

1.1 Технология фрагментированного программирования

Технология фрагментированного программирования (ТФП) – это технология параллельного программирования, целью которой является автоматизация решения задач системного параллельного программирования при реализации численных моделей. Суть ее состоит в следующем [3]:

- Параллельная программа представляется в виде двух отдельных слоев:
  - теоретико-множественное описание алгоритма – переносимое, независимое от архитектуры вычислительной системы;
  - реализационная часть – исполнительная система, обеспечивающая эффективное исполнение алгоритма на конкретной вычислительной системе.
- Алгоритм представляется в специальном фрагментированном виде: в виде множества переменных (фрагментов данных), множества операций (фрагментов вычислений) и отношений на них.

Разделение на фрагменты сохраняется до самого момента исполнения – это позволяет исполнительной системе динамически выполнять настройку на имеющиеся ресурсы. Фрагментированный алгоритм содержит только минимальное управление, определяемое зависимостями по данным, и не содержит распределения ресурсов. Таким образом, он допускает множество способов исполнения, что обеспечивает его переносимость. Задача исполнительной системы – выполнить отображение объектов алгоритма (переменных, операций) на ресурсы конкретной вычислительной системы, автоматически обеспечивая все необходимые динамические свойства параллельной программы [2]. Фрагментация – это технологический

прием, позволяющий уменьшить число объектов алгоритма и тем самым упростить задачу построения эффективного распределения ресурсов и управления.

В ТФП алгоритм – это четвёрка $< X, O, In, Out >$, где

- X – множество переменных (фрагментов данных);
- O – множество операций (фрагментов вычислений);
- $In : X \to O$ – отношение «вход», которое определяет множество входных переменных для каждой операции;
- $Out : X \to O$ – отношение «выход», которое определяет множество выходных переменных для каждой операции.

Таким образом, в базовой модели представления, алгоритм – двудольный ориентированный ациклический граф с вершинами двух видов: переменные единственного присваивания и операции единственного срабатывания (это тождественно определению вычислимой функции Клини). Перед началом работы алгоритма некоторые переменные получают значения. В процессе работы алгоритма каждая операция может сработать независимо от других, если все ее входные переменные получили значения. В результате срабатывания операции значения получают ее выходные переменные. Во избежание неоднозначности вводится следующее правило: две операции не могут одну и ту же переменную в качестве выхода. Перед началом работы алгоритма некоторые переменные получают значения. Исполнение завершается в тот момент, когда не остаётся ни одной операции, которая может сработать. Таким образом, базовая модель представления алгоритма предусматривает его асинхронное исполнение, обеспечивая тем самым возможность отображения алгоритма на любую параллельную архитектуру.

Для представления более сложных численных алгоритмов, базовая модель представления алгоритма расширяется набором средств:

- условные операции – необходимы для представления алгоритмов с ветвлениями,
- структурированные операции (аналог подпрограмм в процедурных языках программирования) – необходимы для иерархического описания алгоритма,
- массивы переменных и операций – необходимы для компактного представления алгоритмов большого и потенциально бесконечного размера.

Различаются два вида массивов в модели представления алгоритма в ТФП:

- Массивы, размер которых известен до начала работы с ними. Будем называть их «массивы определенного размера».

- Массивы, размер которых вычисляется в процессе работы с ними. Будем называть их «массивы неопределенного размера». Работа с такими массивами в программе реализуется с помощью цикла типа while. В ТФП допускаются только одномерные массивы неопределенного размера.

Трансляцией алгоритма из входного формата, отображением его на имеющиеся ресурсы и последующим исполнением занимается специальная система фрагментированного программирования LuNA. Она система состоит из компилятора и подсистемы времени исполнения – runtime-системы. Компилятор принимает статические решения по заданию порядка исполнения операций и частичному распределению ресурсов, например: выбор размера фрагментов, отображение переменных единственного присваивания алгоритма в переменные множественного присваивания, преобразование декларативного описания алгоритма в императивную фрагментированную программу, выбор стратегии начального распределения данных и вычислений. Runtime-система принимает динамические решения: обеспечивает эффективное исполнение фрагментированной программы, передачу данных между узлами мультикомпьютера, совмещение счета с межузловыми обменами (за счет многопоточности), динамическую настройку на ресурсы и балансировку загрузки. Реализуется runtime-система как множество параллельных взаимодействующих процессов.

Таким образом, существенными для дальнейшей работы являются следующие особенности системы фрагментированного программирования:

- представление фрагментов данных и фрагментов вычислений как «черных ящиков» без возможности узнать их внутреннее строение и как-то использовать это знание,

- асинхронное исполнение фрагментов вычислений алгоритма,

- отсутствие монопольного доступа какой-либо части алгоритма к вычислительным ресурсам,

- возможное перемещение фрагментов данных и вычислений в результате динамической балансировки загрузки.

1.2 Операция редукции

Операция редукции является частью многих численных алгоритмов. Входом операции редукции является множество переменных (массив), а выходом – одна переменная. Выделим в численных алгоритмах два вида редукции: полная редукция (над массивом определенного размера) и частичная (над массивом заранее неопределенного размера). В случае полной редукции, число входных значений конечно и заранее известно. Такая операция используется, например, в матрично-векторных операциях и в сеточных задачах. В случае частичной редукции, число входных значений заранее неизвестно и может быть потенциально бесконечным, а результат операции – последовательность частичных результатов. Решение об остановке вычисления редукции программа принимает runtime-система на основе полученных частичных результатов, либо каких-то других факторов. Такая операция, например, используется в задачах, решаемых методом Монте-Карло (поиск среднего значения результатов эксперимента при неизвестном заранее количестве экспериментов).

1.2.1 Формальная модель редукции

Определим формальную модель редукции. Пусть есть произвольное множество S, бинарный оператор $\circ$, определённый на этом множестве, и список A из элементов множества S:

$$\circ : S \times S \to S,$$

$$A = \{a_1, a_2, a_3, ..., a_{n-1}, a_n\};\ a_i \in S.$$

Левая свёртка списка – это функция над списком A и оператором $\circ$, которая определяется следующим образом:

$$fold_l(A, \circ) = ((...((a_1 \circ a_2) \circ a_3) \circ ...) \circ a_{n-1}) \circ a_n.$$

Аналогичным образом определяется правая свёртка:

$$fold_r(A, \circ) = a_1 \circ (a_2 \circ (a_3 \circ (... \circ (a_{n-1} \circ a_n)...))).$$

Операторы, удовлетворяющие свойствам ассоциативности и коммутативности удобны тем, что позволяют вычислять свёртку в произвольном порядке, следовательно список A можно представлять в виде неупорядоченного множества. Для асинхронных параллельных

вычислителей это имеет особое значение, так как такие вычислители эффективно работают, когда алгоритм имеет мало ограничений на порядок исполнения операций. В дальнейшем под операторами, используемыми в редукции, будут подразумеваться ассоциативные коммутативные операторы, если не указано иное. Примеры операторов: сумма, произведение, среднее, максимум, минимум, логическое «и», логическое «или». Будем называть полной редукцией множества A следующую конструкцию:

$$reduce(A, \circ) = a_1 \circ a_2 \circ ... \circ a_n,$$

причем будем считать, что нумерация элементов и порядок срабатывания операторов произвольные.

Пусть есть бесконечное множество B из элементов множества S и бесконечная последовательность вложенных друг в друга подмножеств этого множества:

$$B = \{b_1, b_2, b_3, ...\};\ b_i \in S,$$

$$B_s = \{B_0, B_1, B_2, ...\};\ B_i \subseteq B_{i+1};\ B_i \subset B$$

Тогда результатом частичной редукции будем называть последовательность полных редукций B_i:

$$reduce_p(B_s, \circ) = \{reduce(B_0, \circ), reduce(B_1, \circ), reduce(B_2, \circ), ...\}$$

Одной из наиболее распространенных операций редукции в численных алгоритмах является суммирование чисел с плавающей точкой. Известно, что машинная операция сложения чисел с плавающей точной не обладает свойством ассоциативности, т.е. порядок выполнения операций суммирования влияет на результат. В рамках данной работы возможные проблемы с ассоциативностью оператора $\circ$ не будут рассматриваться. Будем считать, что все проблемы с точностью в случае оператора суммирования вещественных чисел были учтены при реализации этого оператора, например, путем использования арифметики повышенной точности [4].

1.2.2 Модель редукции в ТФП

Система LuNA обеспечивает параллельное асинхронное исполнение фрагментированного алгоритма на различных вычислителях. Эффективное исполнение системных операций,

таких как редукция, должно автоматически обеспечиваться runtime-системой. Таким образом, в runtime-систему должен быть заложен алгоритм выполнения операций редукции, способный настраиваться на текущую конфигурацию вычислительной системы и системную обстановку.

Целевой архитектурой системы фрагментированного программирования является мультикомпьютер, состоящий из многоядерных узлов с общей памятью. На каждом узле работает один процесс runtime-системы. Узлы связаны с помощью коммуникационной подсистемы с заданными характеристиками передачи данных: пропускной способностью и латентностью.

Для распределенной runtime-системы алгоритм редукции представляется как распределенная процедура, которая запускается runtime-системой на подмножестве узлов – вовлечённых узлах. Разные процессы runtime-системы, работающие на разных вовлеченных узлах, передают своим локальным процедурам редукции входные фрагменты данных, в результате чего на целевом узле процедура редукции возвращает runtime-системе результат (или последовательность результатов) редукции. Множество вовлеченных узлов в процессе работы алгоритма будем считать неизменным. Таким образом, будем рассматривать задачу в сокращенной постановке – без учета возможной динамической балансировки загрузки.

Фрагменты данных, поступающиеые на вход операции редукции, будем называть входными. Входные фрагменты данных появляются (вычисляются в ходе работы алгоритма) на вовлеченных узлах, причём порядок появления, время появления и распределение входных фрагментов данных по вычислителям в общем случае не известны.

В случае полной редукции известно общее число входных фрагментов данных. Кроме того, будем считать, что каждый вовлеченный узел в каждый момент работы алгоритма «знает», будут ли еще у него появятся входные фрагменты данных. Целью работы алгоритма редукции является получение значения полной редукции над заданной операцией и множеством входных фрагментов данных на одном из узлов вычислительной системы – так называемом целевом узле. Критерием качества алгоритма здесь является как можно более раннее получение результата полной редукции на целевом узле. Так как существует нижняя граница времени получения результата полной редукции, определяемая временем появления последнего входного фрагмента данных, то целью поиска оптимального алгоритма становится минимизация промежутка времени между появлением последнего входного фрагмента данных на одном из вовлеченных узлов и получением результата полной редукции на целевом узле.

В случае частичной редукции, входные фрагменты данных появляются на вовлеченных узлах до тех пор, пока runtime-система не решит остановить их генерацию, исходя из каких-

то внешних факторов (например, на основании оценки последовательности результатов редукции или по команде пользователя). При этом runtime-система сигнализирует об этом всем вовлеченным узлам, после чего входные фрагменты данных на них больше не появляются. Целью работы алгоритма редукции в этом случае является периодическое получение результатов частичной редукции на целевом узле. Критерием качества алгоритма в случае частичной редукции может быть как можно более быстрый учет появившихся фрагментов данных в частичном результате, т.е. минимизация промежутка времени между появлением очередного входного фрагмента данных и его учетом в очередном результате частичной редукции.

Если дополнительно ввести в модель временные характеристики выполнения оператора $\circ$, то при выборе алгоритма редукции можно учитывать загруженность отдельных узлов.

1.3 Постановка задачи

Целью работы является разработка алгоритма асинхронной распределенной редукции для использования в системе фрагментированного программирования. Для достижения цели ставятся следующие задачи:

- Выделить требования, которым должен удовлетворять алгоритм редукции при его использовании в системе фрагментированного программирования.
- В соответствии с требованиями разработать параметризованный алгоритм асинхронной распределенной редукции, характеристики работы которого определяются значениями параметров.
- Реализовать эмулятор работы алгоритма редукции для оценки его характеристик при различных параметрах.

Глава 2

Обзор алгоритмов редукции

2.1 Реализация редукции в MPI

Самой распространенной на сегодняшний день библиотекой коммуникационных операций для систем с распределенной памятью является библиотека MPI. В стандарт библиотеки MPI входит операция синхронной редукции `MPI_Reduce`. Она выполняется одновременно на группе из N параллельно работающих процессов. Каждый процесс имеет один входной фрагмент данных. В типичном случае использования операции `MPI_Reduce` предполагается, что входной фрагмент данных – вектор длины K значений одного типа: $V_n = \{v_{n,1}, v_{n,2}, ..., v_{n,K}\}$, где n – номер процесса, K - длина вектора, одинаковая для всех. Операция `MPI_Reduce` редуцирует вектора поэлементно и сохраняет результат на одном из процессов, указанном в параметре операции:

$$result = \{r_1, ..., r_K\};\ r_i = v_{1,i} \circ v_{2,i} \circ ... \circ v_{N,i}$$

Оператор $\circ$ задаётся программистом и может быть не коммутативным. В общем случае элементы вектора могут быть разных типов.

В вышедшую в 2012 году версию 3.0 стандарта MPI также был включен неблокирующий аналог операции `MPI_Reduce` – операция `MPI_Ireduce`. Она позволяет использовать операцию редукции в асинхронных программах.

Кроме операций `MPI_Reduce` и `MPI_Ireduce` в стандарте MPI присутствует ряд других редукционных операций (префиксная, с рассылкой элементов вектора-результата, с рассылкой всего результата и др.), а также их асинхронные аналоги. В рамках данного обзора эти операции рассматриваться не будут.

Далее рассмотрим алгоритмы исполнения операций `MPI_Reduce` и `MPI_Ireduce`, ис-

пользующиеся в некоторых реализациях библиотеки MPI.

2.1.1 Алгоритмы редукции в Open MPI

В реализации Open MPI [5] операция `MPI_Ireduce` реализована в виде алгоритма «двоичное дерево» (binary tree).

Алгоритм работает следующим образом. Изначально каждый процесс владеет одним вектором длины K. Процессы нумеруются таким образом, что нулевой процесс является целевым. Ниже представлен алгоритм работы процесса с номером n, где $n \in \{0, ..., N-1\}$, а N – количество процессов.

1. Вводятся переменные i – номер стадии алгоритма и v – промежуточный результат редукции. Переменная i принимает значение 0, переменная v получает значение локального входного фрагмента данных.

2. Если $2^i \geq N$, то перейти к шагу 5.

3. Соседями слева и справа будем называть процессы с номерами $n - 2^i$ и $n + 2^i$ соответственно.

   Если n не делится нацело на 2^{i+1}, то отправить значение v соседу слева и окончить исполнение алгоритма в этом процессе.

   Если n делится нацело на 2^{i+1} и сосед справа существует, то принять значение от соседа справа и сохранить его в переменную v_1. Вычислить $v \circ v_1$ и сохранить результат в переменную v.

4. Вернуться к шагу 2, увеличив значение переменной i на единицу.

5. Если номер узла 0, то значение v на этом узле есть результат выполнения операции редукции. Конец алгоритма.

Направленный граф, в котором вершины соответствуют процессам, а рёбра – факту передачи значения от одного процесса к другому (шаг 3), представляет собой двоичное дерево. Высота этого дерева равна $\lceil log_2 N \rceil$, из чего следует, что время работы ограничено снизу временем последовательной передачи $\lceil log_2 N \rceil$ векторов.

2.1.2 Алгоритмы редукции в MPICH

В работе [6] представлены несколько алгоритмов редукции для реализации в MPI. Алгоритм «двоичное дерево» там признан не оптимальным, поскольку после каждого шага

количество работающих процессов уменьшается вдвое. Вместо него автором было предложено несколько вариаций алгоритма операции `MPI_Reduce`, состоящих из двух этапов: «Reduce_scatter» и «Gather».

1. На этапе «Reduce_scatter» входные вектора разбиваются на подвектора. Подвектора распределяются между процессами таким образом, что каждый процесс имеет достаточно данных для того, чтобы получить точное значение одного из подвекторов результирующего вектора. Основной принцип первого этапа заключается в том, что на каждом шаге этого этапа, каждый процесс делит каждый свой вектор на две половины. Одну половину он оставляет себе, а второй обменивается с соседом. Таким образом, на первом шаге процесс имеет целый вектор, на втором – два полувектора, на третьем – четыре четвертьвектора и так далее, пока процесс не будет иметь N подвекторов, где N – число процессов, равное числу изначальных векторов. Затем процесс локально выполняет редукцию всех своих подвекторов, получив подвектор результирующего вектора.

2. На этапе «Gather» подвектора результирующего вектора, полученные на предыдущем этапе, собираются в одном из процессов.

Описанные выше этапы работы алгоритма подразумевают, что N является степенью двойки. В том случае, если это не так, одна из вариаций алгоритма, названная «разделение пополам и сдваивание» (halving&doubling), использует только $2^{\lfloor log_2 N \rfloor}$ процессов, а другая, названная «двоичные блоки» (binary blocks), разбивает множество процессов на несколько блоков размером со степени двойки и исполняет этапы на каждом блоке отдельно.

В реализации MPICH [7] используются алгоритмы «разделение пополам и сдваивание» и «двоичное дерево». Кроме того, для случая, когда несколько процессов оказываются на одном узле мультикомпьютера, в MPICH реализован предварительный сбор частичного результата на узле.

2.2 Реализация редукции в Charm++

Система программирования Charm++ предназначена для создания асинхронных параллельных программ для систем с общей и распределенной памятью [8]. Лежащие в ее основе принципы очень близки принципам ТФП, поэтому она была включена в обзор.

Charm++ включает язык (расширение языков C++, Fortran 90), компилятор, runtime-систему, а также множество подключаемых библиотечных модулей, реализующих системные функ-

ции, такие как различные стратегии коллективных обменов данными (рассылка, редукция), алгоритмы динамической балансировки загрузки, работа с контрольными точками. Программа на Charm++ представляет собой распределенное множество объектов, называемых chare. Объекты содержат в себе и данные, и код. Взаимодействуют объекты между собой посредством передачи сообщений, инициирующих запуск соответствующих методов у целевых объектов.

Операции редукции в Charm++ определены над группами объектов. После инициализации операции редукции каждый объект группы асинхронно вызывает специальный метод `contribute`, задающий вклад данного объекта в операцию редукции. Когда все объекты группы сделали свой вклад, вызывается заранее заданный метод объекта, инициировавшего редукцию, которому в качестве параметра передается результат редукции.

В Charm++ используется следующий алгоритм редукции. Во первых, при создании группы объектов для соответствующего множества узлов мультикомпьютера строится остовное дерево заданной степени. Степень остовного дерева задается в качестве параметра при создании группы (по умолчанию дерево двоичное). Во время работы алгоритма редукции частичный результат собирается на каждом узле мультикомпьютера со всех локальных объектов, участвующих в редукции. Далее, сбор результатов на целевом узле осуществляется по построенному остовному дереву.

Редукция в Charm++ может происходить даже во время выполнения динамической балансировки загрузки. При этом эффективность выполнения операции редукции падает до того момента, как остовное будет автоматически перестроено.

2.3 Выводы

Модель редукции в MPI имеет следующие особенности:

- Каждый процесс, участвующий в редукции, дает только один вклад в редукцию, причем, процессы жестко распределяются по ядрам/узлам мультикомпьютера.
- Редуцируемые данные имеют явное представление в виде векторов элементов.
- Поддерживаются только операции полной редукции.

Все эти особенности активно используются для оптимизации алгоритмов редукции в MPI. Так как для системы фрагментировнного программирования перечисленные свойства, как

правило, не выполняются, то алгоритмы, используемые в различных реализациях MPI, имеют ограниченное применение в системе фрагментированного программирования.

Модель редукции в системе Charm++ во многом совпадает с моделью редукции в ТФП: та же модель вычислителя, произвольное распределение объектов по ресурсам, асинхронное исполнение фрагментов вычислений, фрагменты для всей системы представляются как черные ящики. Алгоритмы, использующиеся для реализации редукции в Charm++, могут быть применены и для реализации полной редукции в системе фрагментированного программирования. Однако Charm++ не поддерживает операцию асинхронной частичной редукции, а только последовательность полных редукций, требующих на каждом этапе участия всех вовлеченных объектов.

Глава 3

Разработка алгоритмов параллельной асинхронной редукции

3.1 Идеи алгоритма

Процесс вычисления операции редукции можно представить в виде бинарного дерева, в котором листья содержат значения входных фрагментов данных, а корневая вершина каждого поддерева содержит результат полной редукции всех листьев этого поддерева. Будем называть это дерево деревом редукции. Корень дерева редукции содержит результат полной редукции всех входных фрагментов данных алгоритма. Корень некоторого поддерева редукции содержит частичный результат.

Каждая вершина дерева редукции физически расположена на каком-либо вовлеченном узле. Листья расположены на тех узлах, на которые они поступили, остальные вершины дерева – на том узле, на котором была выполнена операция, вычисляющая их значение. Дуги между вершинами дерева означают либо передачу значения фрагмента данных от узла потомка к узлу предка, если это разные узлы, либо то, что значение фрагмента остаётся на этом же узле в противном случае.

Для построения дерева редукции необходима информация о распределении событий появления входных фрагментов данных по времени и по вовлеченным узлам. Так как из-за динамики работы runtime-системы эта информация становится доступна только в процессе исполнения, то для построения алгоритма редукции использовать дерево редукции не представляется возможным.

Вместо этого было решено спланировать оптимальный путь движения входных фрагментов данных и частичных результатов между вовлеченными узлами к целевому. Очевидно,

что все множество путей должно формировать остовное дерево на множестве вовлеченных узлов с корнем в целевом узле. Будем называть его деревом вычислителя.

Таким образом, разработка алгоритма редукции разбивается на два этапа:

1. построение дерева вычислителя,
2. разработка алгоритма работы узла в вершине дерева.

3.2 Построение дерева вычислителя

Дерево вычислителя представляет собой объединение путей следования фрагментов данных от узла, на которых они появились (были получены от runtime-системы), до целевого узла. В вершинах дерева происходит частичная редукция фрагментов данных, после чего частичный результат продолжает путь вверх по дереву.

В рамках работы будем считать, что дерево вычислителя строится один раз до запуска алгоритма редукции и не меняется в процессе работы алгоритма.

Дерево вычислителя – это глобальная структура, для построения которой необходимо учесть параметры всех уровней иерархической вычислительной системы. Однако, централизованный алгоритм построения дерева нежелателен в распределенной вычислительной системе большого размера. Лучшим вариантом будет использовать детерминированный алгоритм, который сможет построить дерево на изначально доступной всем узлам информации. Такой информацией может быть статическая информация о всей вычислительной системе (топология, технические характеристики узлов и каналов связи) и множество вовлечённых узлов. Обладая этой информацией каждый вовлеченный узел должен однозначно определить своих соседей в дереве вычислителя.

Соседи каждого узла в дереве делятся на две группы:

- Узлы, расположенные по дереву дальше от корня – будем называть их потомки.
- Один узел, расположенный по дереву ближе к корню – будем называть его предок.

Высота дерева вычислителя определяет максимальную длину пути, который необходимо преодолеть фрагментам данных чтобы добраться до целевого узла. Поэтому целесообразно строить дерево не очень большой высоты. С другой стороны, уменьшение высоты дерева влечёт за собой увеличение количества потомков у всех нелистовых вершин, что увеличивает нагрузку на эти вершины: узел может не «успевать» принимать данные или выполнять операции, если потомки будут отправлять слишком много фрагментов данных в единицу

времени. Поэтому лучшим в каждом конкретном случае будет некоторый компромиссный вариант.

Определить оптимальную оптимальную степень вершины до запуска алгоритма редукции может оказаться невозможным вследствие непредсказуемой динамики исполнения. Однако, к решению этого вопроса можно приблизиться с помощью профилирования процесса исполнения и корректировки высоты дерева от запуска к запуску на основе полученной информации.

Для однозначного построения остовного дерева заданной степени k на вовлеченных узлах может быть использован следующий алгоритм, основанный на построении дерева в ширину.

1. Пусть все вовлеченные узлы пронумерованы без повторений (не обязательно подряд). Организовать список номеров вовлеченных узлов в порядке возрастания.

2. В качестве корня дерева взять целевой узел. Исключить его номер из списка.

3. Взять из списка k минимальных номеров, сделать соответствующие узлы потомками корня. Исключить их номера из списка.

4. Выбирая по порядку номера из списка, добавлять вершинам текущего максимального уровня потомков следующего уровня, пока у всех не будет по k потомков, или пока список не исчерпается. Добавлять следует в таком порядке: сначала всем раздать по одному потомку, затем по второму, и т.д. до k-х.

5. Если список закончился, то конец. Иначе выполняем шаг 4 для следующего уровня.

Имея всю входную информацию (список вовлеченных узлов, номер целевого узла, свой номер) и следуя этому алгоритму, каждый узел может определить свое место в дереве без дополнительных коммуникаций.

3.3 Алгоритм работы узла

Задача каждого узла в процессе работы алгоритма редукции состоит в следующем:

- Принимать входные фрагменты данных от локального процесса runtime-системы.

- Принимать фрагменты данных, содержащий частичные результаты редукции, от потомков.

- Выполнять редукцию принимаемых фрагментов данных.
- В случае, если узел является не целевым, отправлять сообщения предку. Если узел является целевым, то вернуть результат редукции runtime-системе.
- Принимать сигнал runtime-системы о том, что входных фрагментов на данный узел больше поступать не будет.

Далее рассмотрим отдельно алгоритм работы узла для случаев полной и частичной редукции.

3.3.1 Алгоритм для полной редукции

В случае операции полной редукции наиболее оптимальным алгоритмом работы узла будет выполнить редукцию всех локальных фрагментов данных, а затем передать этот частичный результат вверх по дереву. Кроме того, нужно правильно отработать завершение работы алгоритма.

Полностью алгоритм поведения узла в случае полной редукции выглядит следующим образом:

1. Вводится счетчик завершений, изначально равный нулю, и переменная «локальный результат», значение которой сначала не определено.
2. При первом поступлении фрагмента данных (входного или от потомков) он становится «локальным результатом».
3. При последующих поступлениях фрагментов данных они сразу же редуцируются с «локальным результатом», образуя новый «локальный результат».
4. При получении локального сигнала о завершении или сигнала от одного из потомков счетчик завершений увеличивается на 1.

   Если значение счетчика завершений достигло величины $\textit{число потомков}+1$, то:

   (a) Если узел целевой, то отправить «локальный результат» runtime-системе в качестве результата операции полной редукции. Конец алгоритма.

   (b) Если узел не целевой, то отправить предку «локальный результат» (если его значение определено), а затем сигнал о завершении. Конец алгоритма.

Заметим, что для корректной работы алгоритма должно быть обеспечено сохранение порядка передаваемых между узлами сообщений.

3.3.2 Алгоритм для частичной редукции

В случае операции частичной редукции целевой узел должен периодически выдавать runtime-системе как можно более актуальный результат частичной редукции. Для поддержания актуальности результата на целевом узле все остальные вовлеченный узлы должны периодически отправлять предкам свои текущие частичные результаты.

Возникает вопрос, как часто узлы должны отправлять свои текущие результаты? Если делать это слишком часто, то возможна слишком большая нагрузка на коммуникационную подсистему, в результате чего будут иметь место неоправданные задержки. Кроме того, задержки могут возникать вследствие загрузки узлов-предков в результате выполнения большого числа операций редукции. Однако, если выполнять передачи слишком редко, то результат на целевом узле будет менее актуальным.

Таким образом, требуется ввести параметр алгоритма: T_{wait} – интервал времени между соседними отправками узла предку частичного результата редукции. При различных характеристиках системного окружения у этого параметра будут различные оптимальные значения.

В отличие от алгоритма полной редукции, в алгоритме частичной редукции при поступлении сигнала о завершении может быть целесообразно немедленно закончить работу, не дожидаясь учета всех выданных runtime-системой входных фрагментов данных. Единственное, что в данном случае необходимо обеспечить, так это чтобы после завершения работы алгоритма не оставалось никем сообщений. Поэтому для корректного завершения алгоритма частичной редукции также используется счетчик завершений.

Алгоритм поведения узла в случае частичной редукции выглядит следующим образом:

1. Вводятся:

   (a) счетчик завершений, изначально равный нулю;

   (b) таймер, изначально установленный в значение T_{wait} и убывающий с течением времени;

   (c) переменная «локальный результат», значение которой сначала не определено.

2. При поступлении фрагмента данных (входного или от потомков) и при условии, что значение счетчика завершений равно нулю:

   (a) Если значение переменной «локальный результат» не определено, то поступивший фрагмент данных становится «локальным результатом».

(b) Иначе: поступивший фрагмент данных редуцируется с «локальным результатом», получая новый «локальный результат».

3. При достижении таймером нулевого значения и при условии, что значение счетчика завершений равно нулю:

   - Таймер заново устанавливается в значение T_{wait}.
   - Если значение переменной «локальный результат» определено, то оно отсылается предку, а затем становится не определенным.

4. При получении локального сигнала о завершении или сигнала от одного из потомков счетчик завершений увеличивается на 1.

   Если значение счетчика завершений достигло величины *число потомков*$+1$, то:

   (a) Если узел не целевой, то отправить предку сигнал о завершении.

   (b) Конец алгоритма.

Если в каком-то случае потребуется учесть все выданные runtime-системой входные фрагменты данных, то для него алгоритм частичной редукции может быть легко модифицирован. Такой алгоритм будет полностью повторять алгоритм для полной редукции с добавлением таймера и соответствующей реакции на него.

Глава 4

Оценка алгоритмов

4.1 Эмулятор работы алгоритмов редукции

Для определения оптимальных параметров разработанных алгоритмов редукции был разработан эмулятор на языке Python, моделирующий работу этих алгоритмов на распределенной вычислительной системе. В эмуляторе используется тактированное модельное время.

На вход эмулятор получает набор параметров:

- параметры вычислительной системы,
- параметры алгоритма,
- параметры задачи.

В качестве результата эмулятор возвращает оценки:

- Для алгоритма полной редукции, значение T_{res} – суммарное время работы алгоритма.
- Для алгоритма частичной редукции: график значения числа $\alpha(1000)$ в зависимости от времени, где $\alpha(n)$ - среднее время запаздывания последних n учтённых фрагментов, а время запаздывания – промежуток времени между генерацией фрагмента и его учётом в очередном результате частичной редукции.

В качестве параметров вычислительной системы были выбраны:

- N – число вовлеченных узлов вычислительной системы,
- T_{lat} – латентность коммуникационной среды,
- T_{bth} – время передачи по сети одного фрагмента данных,

- T_{op} – время выполнения локальной операции редукции над двумя операндами.

Параметры коммуникационной среды интерпретируются следующим образом: если один узел отправляет другому фрагмент данных, то этот фрагмент появляется у получателя через модельное время T_{lat}, после этого получатель должен потратить T_{bth} тактов для получения этого фрагмента. Таким образом, если один узел отправляет другому K фрагментов данных, то на узле назначения K-й фрагмент появятся через модельное время $T_{lat} + K \times T_{bth}$.

В качестве параметров алгоритма были использованы:

- P – степень дерева вычислителя (максимальное число потомков узла).
- T_{wait} – период времени между отправками частичного результата (для операции частичной редукции).

Дерево вычислителя с заданным числом вершин N и заданной степени P генерируется автоматически с помощью алгоритма, представленного в разделе 3.2.

Параметры задачи:

- N_F – число генерируемых фрагментов данных на узел (для операции полной редукции),
- T_{Gmin}, T_{Gmax} – параметры темпа генерации входных фрагментов данных: минимальное и максимальное время в тактах между появлениями входных фрагментов данных на каждом узле.

Фрагменты данных генерируются на каждом узле через равномерно распределенные случайные промежутки времени в диапазоне от T_{Gmin} до T_{Gmax}.

4.2 Исследование параметров алгоритмов

4.2.1 Полная редукция

Для теста работы алгоритма полной редукции были выбраны следующие параметры:

- Параметры вычислительной системы: число узлов $N = 31$, $T_{op} = 10$, $T_{bth} = 10$, T_{lat} – варьировалась.
- Параметры генерации: $N_F = 100$, $T_{Gmin} = 10$, $T_{Gmax} = 11$.

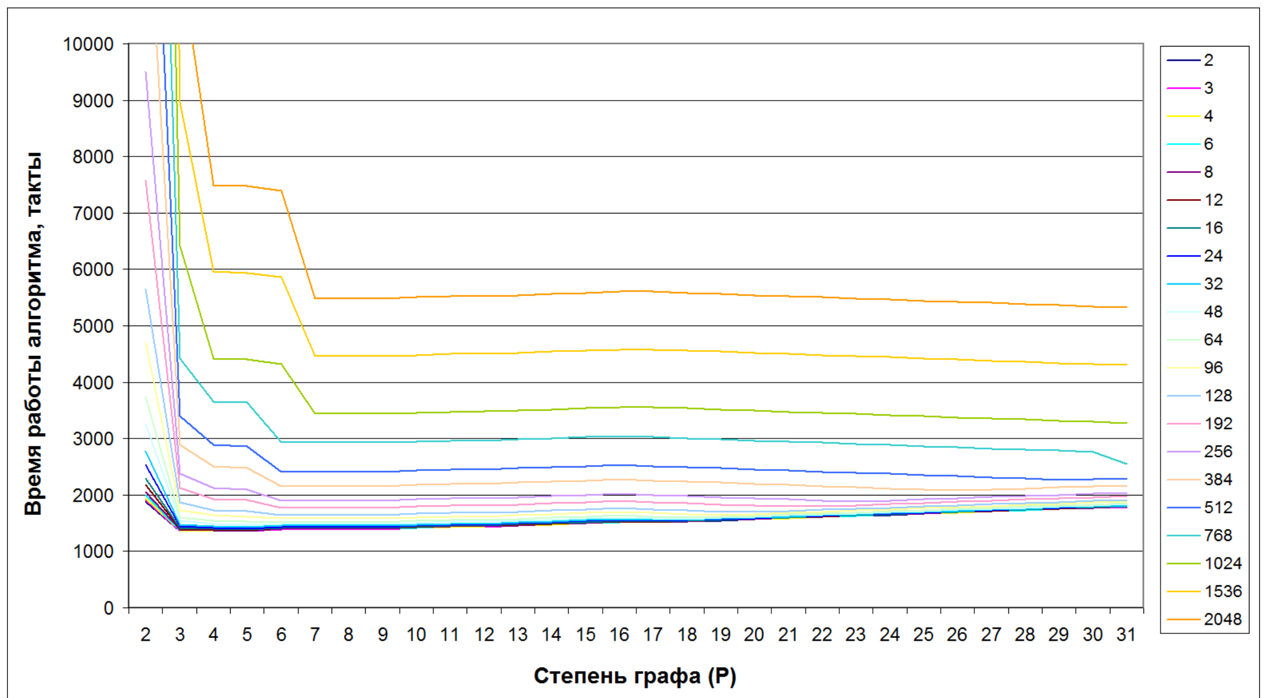


Рис. 4.1: Зависимость времени работы алгоритма полной редукции от степени дерева вычислителя и латентности

На рис. 4.1 представлена зависимость времени работы алгоритма полной редукции от степени дерева вычислителя и латентности. Различные графики приведены для различных величин латентности (чем больше величина латентности, тем выше располагается график). По графику видно, что для каждой величины латентности существует оптимальное значение степени дерева вычислителя. На рис. 4.2 представлена зависимость оптимальной степени дерева вычислителя от величины латентности.

4.2.2 Частичная редукция

Для теста работы алгоритма полной редукции были выбраны следующие параметры:

- Параметры вычислительной системы: число узлов $N = 32$, $T_{op} = 10$, $T_{bth} = 10$, $T_{lat} = 10$.
- Параметры генерации: $T_{Gmin} = 20$, $T_{Gmax} = 80$.

На рис. 4.3(а) видно, что при достаточно высоком значении параметра P и низком значении параметра T_{wait}, алгоритм не справляется и значение $\alpha(1000)$ растёт со временем. Такой результат назовём нестабильным.

С другой стороны, на рис. 4.3(б) видно, что при низких значениях параметра P и высоких значениях T_{wait}, значение $\alpha(1000)$ колеблется в некотором районе. Чем ниже P и выше T_{wait}, тем выше район колебаний. Такой результат назовём стабильным.

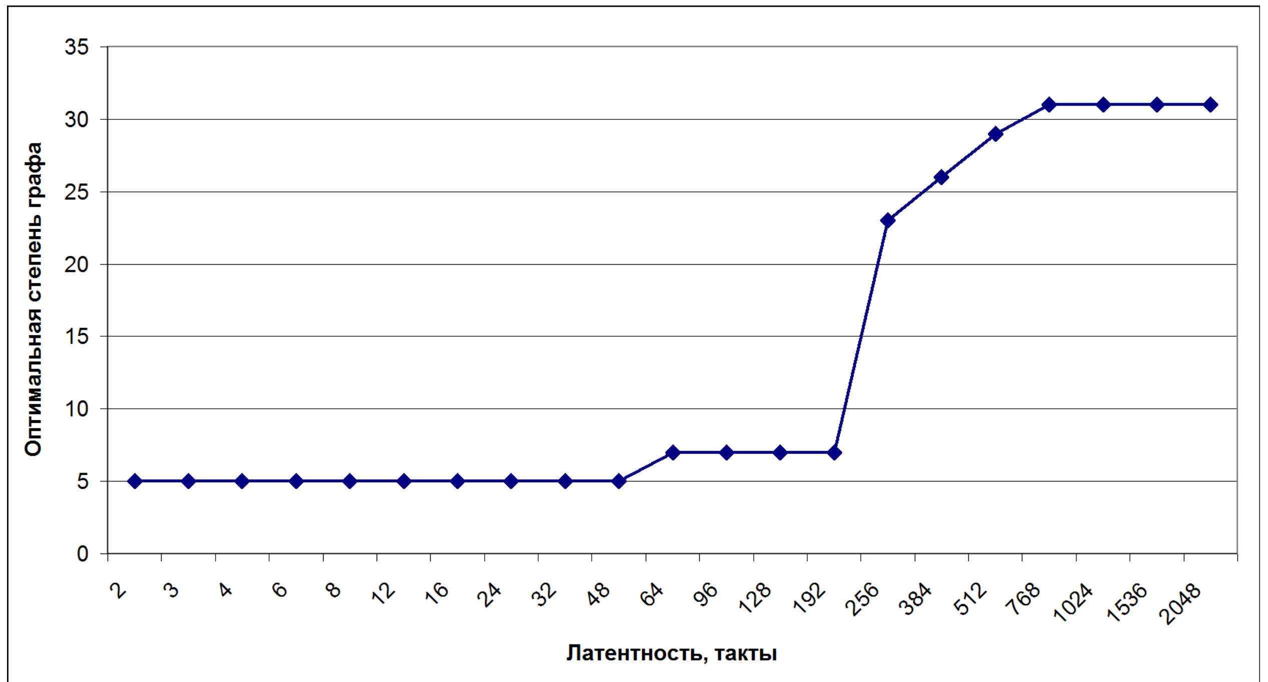
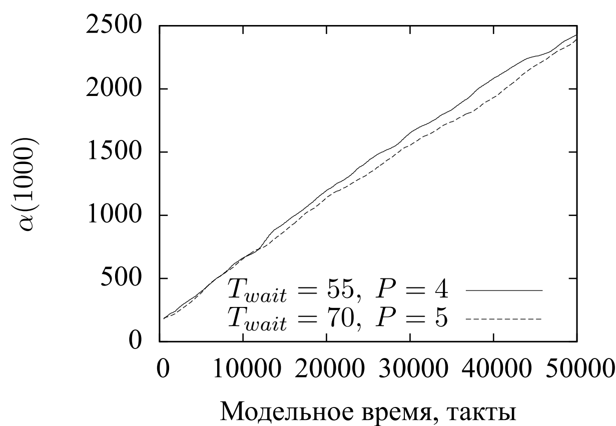
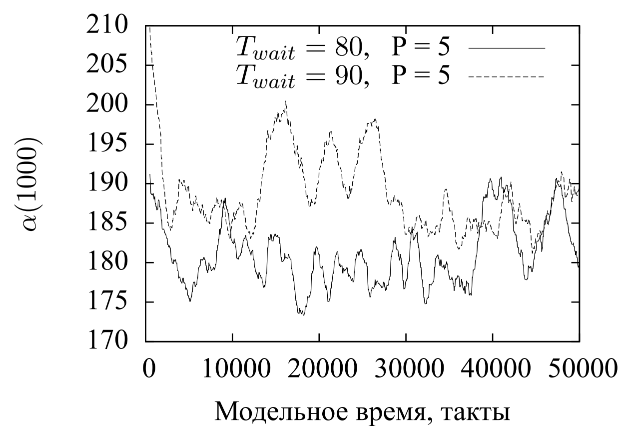


Рис. 4.2: Зависимость оптимальной степени дерева вычислителя от латентности



(а) Нестабильные результаты



(б) Стабильные результаты

Рис. 4.3: Зависимость $\alpha(1000)$ от времени

Заключение

В работе была рассмотрена технология фрагментированного программирования как подход к автоматизации конструирования параллельных программ. Были выделены особенности системы фрагментированного программирования, существенные для реализации системных алгоритмов, в частности, алгоритмов редукции. Выполнен обзор алгоритмов параллельной редукции, использующиеся в существующих системах параллельного программирования. Разработаны два параметризованных алгоритма асинхронной распределенной редукции для операций полной и частичной редукции. Алгоритмы могут быть использованы в системе фрагментированного программирования LuNA. Разработан эмулятор, моделирующий работу алгоритма редукции при заданных характеристиках системного окружения. С помощью моделирования была определена зависимость оптимальных параметров алгоритма редукции от характеристик системного окружения. Основные результаты работы:

- алгоритм асинхронной распределенной редукции для использования в системе фрагментированного программирования,
- эмулятор, моделирующий работу разработанного алгоритма при заданных характеристиках системного окружения.

Направления дальнейшей работы:

- Тестирование алгоритма на реальных вычислительных системах, и определение для них оптимальных параметров.
- Расширение алгоритма для учета динамической балансировки загрузки.
- Разработка асинхронной редукционной операции суммирования вещественных чисел с заданной точностью.

Литература

1. В.А.Вальковский, В.Э.Малышкин. Синтез параллельных программ и систем на вычислительных моделях. – Наука, Новосибирск, 1988, 128 стр.

2. Malyshkin V.E., Perepelkin V.A. Optimization of Parallel Execution of Numerical Programs in LuNA Fragmented Programming System // MTPP-2010 revised selected papers, Springer Berlin Heidelberg, LNCS 6083, 2010. P. 1-10.

3. S.Kireev, V.Malyshkin Fragmentation of Numerical Algorithms for Parallel Subroutines Library // The Journal of Supercomputing. Vol. 57. Number 2. 2011. P. 161-171.

4. Takeshi Ogita, Siegfried M. Rump, Shin'ichi Oishi, Accurate Sum and Dot Product // SIAM Journal on Scientific Computing archive, Volume 26 Issue 6, 2005. P. 1955-1988.

5. Open MPI, http://www.open-mpi.org [Электронный ресурс].

6. Rolf Rabenseifner, Optimization of Collective Reduction Operations // Computational Science - ICCS 2004, Springer Berlin Heidelberg, LNCS 3036, 2004. P. 1-9.

7. MPICH, http://www.mpich.org [Электронный ресурс].

8. Charm++, http://charm.cs.uiuc.edu [Электронный ресурс].