МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 10

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

МАКС ПРЕСС

МОСКВА – 2009

УДК 519.6:517.958
ББК 22.19
П78

Печатается по решению
Редакционно-издательского совета факультета
вычислительной математики и кибернетики МГУ имени М.В. Ломоносова

П78 **Программные системы и инструменты:** Тематический сборник / Под общей ред. Королева Л.Н. – М: Издательский отдел факультета ВМК МГУ (лицензия ИД №05899 от 24.09.2001г.); МАКС Пресс, 2009. – № 10. – 156 с.
ISBN 978-5-89407-411-5
ISBN 978-5-317-03092-6

В данный сборник включены научные работы и сообщения, связанные с общими вопросами программирования и информатики, касающиеся некоторых вопросов математической лингвистики, а также с описанием некоторых инструментальных систем, которые могут оказаться полезными во многих практических приложениях.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненных учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2009 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6:517.958
ББК 22.19

ISBN 978-5-89407-411-5
ISBN 978-5-317-03092-6

СОДЕРЖАНИЕ

От редколлегии:

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные некоторым вопросам общего характера, в частности, в статьях Брусенцова Н. П. и Владимировой Ю. С. рассматриваются проблемы, связанные с анализом классической логики на предмет возможности её использования в моделировании правдоподобных (естественных) рассуждений. Несколько статей посвящены актуальной тематике, связанной с проблемами параллельного программирования для высокопроизводительных комплексов. Ряд статей посвящен теории формальных языков и обработке текстовой информации. В статье Рябова Г. Г. приведен теоретически и практически важный результат по способу кодирования объектов на пространственных решетках. Сборник включает также статьи с описаниями инструментальных программных систем, которые могут представить интерес для специалистов. В нем также публикуются статьи, касающиеся разделов информатики, тесно связанных с программированием.

Раздел I
Общие вопросы программирования и информатики

Брусенцов Н.П.

Отчего классическая логика лишена необходимого следования

Известно, что отношение следования в общепринятой "классической" логике парадоксально: из несуществующего следует все что угодно, общезначимое следует из чего угодно. Именно поэтому, будучи совершенным средством классификации, логика не обеспечивает достоверности умозаключений [1]. Настойчивые попытки устранить парадоксы неизменно оказываются безуспешными. Логика остается неадекватной действительности.

Заблуждение в том, что, несправедливо приписывая несовершенную логику Аристотелю, стремятся создавать "неаристотелевы" логики. Однако как раз в аристотелевой силлогистике следование, выраженное общеутвердительной посылкой "Все x суть y" ("Из x необходимо следует y"), непарадоксально. Непредставимость силлогистики в "классической" логике явно указывает на неполноценность последней, о которой наглядно свидетельствует ее парадоксальность.

Парадоксы обусловлены принятием античными стоиками антиаристотелевского "закона исключенного третьего", превратившего адекватную трехзначную логику Аристотеля в неполноценную двухзначную схоластику. В условиях двухзначности парадоксы неустранимы, однако из известных трехзначных логик непарадоксальна только аристотелева.

Ясно, что надлежит не изобретать неаристотелевы логики, а отменив нелепый "закон исключенного третьего", воссоздать диалектическую логику Аристотеля, использовав современную алгебру и трехзначные диаграммы Льюиса Кэррола [2], обеспечивающие возможность неискаженного отображения реальных отношений.

Принципиальная особенность логики Аристотеля, несоблюденная в других трехзначных логиках (в частности и в логике самого Кэррола), состоит в непременности сосуществования присущности и антиприсущности вещам рассматриваемых качеств. Дело в том, что сущность качества постижима лишь путем сопоставления вещей, которым оно присуще, с вещами, которым оно антиприсуще. Например, качество x реально только при

сосуществовании x-вещей и x'-вещей (штрих символизирует антиприсущность). Универсум Аристотеля УА есть сосуществование присущности и антиприсущности всех рассматриваемых в нем качеств: $VxVx'VyVy'VzVz'$...

Строгая импликация Льюиса $V'xy'$, призванная преодолеть парадоксы материальной импликации $(x \to y) \equiv (x' \vee y)$, в классической логике тождественна $x \to y$, поскольку удовлетворяется при $V'x$ независимо от y и при $V'y'$ независимо от x. Непарадоксальным следованием $(x \Rightarrow y) \equiv V'xy'VxVy'$ она становится в универсуме Аристотеля УА, в котором как x, так и y' непременно существуют.

Отношение необходимого следования определено Аристотелем в "Первой аналитике" [3, стр.215]. Оно контрапозитивно: $(x \Rightarrow y) \equiv (y' \Rightarrow x')$, необходимо не удовлетворяется при $(x \Rightarrow y') \equiv (y \Rightarrow x')$ и не необходимо удовлетворяется/не удовлетворяется, если не соблюдено ни первое, ни второе, в чем и состоит исключенное в двухзначной логике третье, без которого непарадоксальная логика невозможна [4]

Литература

1. Уолтерс Д. "Книга Великой Тайны". Забытое дополнение к "Книге Перемен". - К. "София", М.: ИД "Гелиос", 2002.
2. Кэррол Л. Символическая логика // Л.Кэррол. История с узелками. - М.: "Мир", 1973.
3. Аристотель Сочинения в четырех томах. М.: "Мысль", т.2, 1978.
4. Брусенцов Н.П. Исчерпывающее решение "неодолимой" проблемы парадоксов. М.: Фонд "Новое тысячелетие", 2008

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 20 апреля 2009 г.

Владимирова Ю.С.

Программная реализация аристотелевой силлогистики

Компьютеризация достоверного рассуждения достигается программной реализацией силлогистики Аристотеля, в силу того, что в последней имеется адекватное действительности выражение отношения содержательного следования в виде общеутвердительного суждения $Axy \equiv$ «Всякое x суть y» [1]. В силлогистике терминами x и y обозначают качества, наличием либо отсутствием которых характеризуются рассматриваемые вещи. Каждая вещь представляется совокупностью своих особенностей: xy-вещь обладает особенностями x и y, xy'-вещь – особенностями x и y' (y' – отсутствие качества y). Непарадоксальность представленных в силлогистике отношений обеспечивается подчинением всех силлогистических суждений диалектическому принципу сосуществования противоположностей [2].

Наиболее подходящим алгебраическим представлением суждений аристотелевой силлогистики оказывается их выражение суждениями существования вещей, восходящее к методу диаграмм Льюиса Кэрролла [3]. В качестве символа существования вещи используется префиксный знак дизъюнкции V, несуществования – V', а вещи, о существовании которых ничего не указывается, считаются привходящими. Отношение необходимого следования в терминах суждений существования вещей выражается следующим образом:

$$(x \Rightarrow y) \equiv Axy \equiv \mathrm{V}'xy'\mathrm{V}xy\mathrm{V}x'y'.$$

Принцип сосуществования противоположностей, имеющее представление:

$$\mathrm{V}x\mathrm{V}x'\mathrm{V}y\mathrm{V}y' \ldots .$$

Универсум, удовлетворяющий данному условию, получил название универсума Аристотеля УА [2].

Силлогистика Аристотеля программно реализуется кодированием силлогистических посылок четырех- и восьмитритными векторами, тритам которых сопоставляются суждения существования всевозможных вещей, наиболее полно охарактеризованных в данном универсуме. Так в двухтерминном универсуме имеются четыре особенности, уточнение которых в рамках данного универсума далее невозможно: xy, xy', $x'y$, $x'y'$. Соответственно в данном универсуме силлогистические суждения кодируются четырехтритными векторами. Тритам вектора позиционно сопоставляются указанные четыре особенности. Значениям тритов отвечают статусы существования охарактеризованных этими особенностями вещей: “+” означает

существование соответствующей вещи, "–" – несуществование, "0" – возможность существования. Например, следование y из x $(x \Rightarrow y) \equiv \mathrm{V}xy\mathrm{V}'xy'\mathrm{V}x'y'$ отображается значением четырехтритного вектора (+–0+):

xy	xy'	$x'y$	$x'y'$
+	–	0	+

В трехтерминном универсуме для кодирования силлогистических суждений необходимы восьмитритные векторы, например, $(x \Rightarrow y)_{xyz} \equiv$ (++--00++):

xyz	xyz'	$xy'z$	$xy'z'$	$x'yz$	$x'yz'$	$x'y'z$	$x'y'z'$
+	+	–	–	0	0	+	+

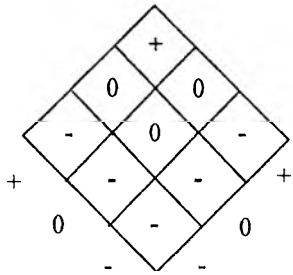
Силлогистический вывод осуществляется приведением исходных посылок в единый универсум, их конъюнктивным совмещением и последующим элиминированием среднего термина, что по существу равнозначно выявлению суждений, сущность которых содержится в совокупности сущностей исходных посылок – в полном соответствии аристотелевскому пониманию необходимого следования. Указанные действия реализуются применением простейших операций над тритными векторами. Приведение посылок в единый трехтерминный универсум достигается удвоением исходного вектора, при котором триты, отличающиеся лишь средним термином имеют одинаковые значения, согласно правилам:

$\mathrm{V}xy \equiv \mathrm{V}xy'z \vee \mathrm{V}xy'z,$

$\mathrm{V}'xy \equiv \mathrm{V}'xy'z\ \mathrm{V}'xy'z,$

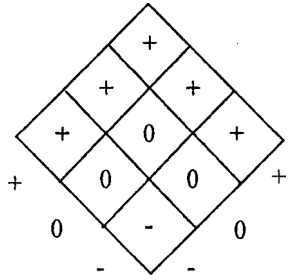
где в качестве среднего взят термин y. В получаемом восьмитритном векторе значение "+" понимается иным, нежели в исходных четырехтритных векторах, образом: "+" соответствует существование xz-вещей, при этом безразлично, будут ли они обладать особенностью y или нет, в то время как "-" по-прежнему означает несуществование xy-вещей, т.е. невозможность как xyz-, так и $xy'z$-вещей. Данная особенность наряду с необходимостью соблюдения принципа сосуществования противоположностей $\mathrm{V}x\mathrm{V}x'\mathrm{V}y\mathrm{V}y'\mathrm{V}z\mathrm{V}z'$ в трехтерминном универсуме учитывается в реализации дальнейших преобразований.

Совмещение исходных посылок осуществляется потритной конъюнкцией отображающих их векторов:



Конъюнкция тритов

Элиминирование среднего термина – объединением тритов полученного восьмитритного вектора в пары таким образом, чтобы особенности, соответствующие тритам одной пары, различались одним только средним термином: (*xyz*, *xy'z*), (*xyz'*, *xy'z'*), … и применением к каждой паре тритов операции дизъюнкции:



Дизъюнкция тритов

Например, решение модуса *Barbara* производится следующим образом:

$$A{xy}A{yz} \equiv (+0\text{-}+)_{xy}(+0\text{-}+)_{yz} \equiv (00{+}{+}\text{--}00)_{xyz}(+0\text{-}{+}{+}0\text{-}+)_{xyz} \equiv$$
$$\equiv (+0\text{-}0\text{---}+)_{xyz} \Rightarrow (+0\text{-}+)_{xz} \equiv V'xz' \equiv Axz.$$

Полученная компьютерная система, реализующая категорическую силлогистику, различает 128 правильных модусов. «Классический» набор силлогистических суждений оказался пополненным непротиворечащими принципу сосуществования противоположностей отношениями эквивалентности и неэквивалентности вещей, например:

$$A{xy}A{yx} \equiv (V'xy'VxyVx'y')(\,V'x'yVxyVx'y') \equiv$$
$$\equiv (+0\text{-}+)xy(+\text{-}0+)_{xy} \equiv (+\text{--}+)_{xy} \equiv (V'xy'V'x'yVxyVx'y') \equiv (x \Leftrightarrow y).$$

Приведенные принцип содержательности суждений и способ получения умозаключений не ограниченны количеством рассматриваемых терминов, что означает возможность выявления достоверных отношений между вещами, охарактеризованными в произвольном УА.

Литература

1. Аристотель Сочинения в четырех томах. – М.: «Мысль», т. 1 – 1975, т. 2 – 1978.
2. Брусенцов Н.П. Парадоксы логики, здравый смысл и диалектический постулат Гераклита-Аристотеля. // Программные системы и инструменты: Тематический сборник факультета ВМиК МГУ им. Ломоносова: № 4 Под ред. Л.Н. Королева. – М.: Издательский отдел ВМиК МГУ, 2003. С. 35-38.
3. Кэррол Л. Символическая логика // Льюис Кэррол. История с узелками. – М.: «Мир», 1973.
4. Брусенцов Н.П. Исчерпывающее решение «неодолимой» проблемы парадоксов. – М., Фонд «Новое тысячелетие», 2008. – 8 с.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 20 апреля 2009 г.

Рябов Г.Г.

Алгебраическое представление кубических структур и супервычисления

(НИВЦ МГУ)

Общеизвестна тенденция, которую можно грубо сформулировать так. То, что вчера было новым в архитектуре суперкомпьютеров, сегодня найдет свое отражение в архитектуре серверов и персональных компьютеров. И с другой стороны, то что освоено электронной технологией и завоевало массовый рынок (рабочие станции и PC), завтра будет массово использоваться при создании суперкомпьютеров.

Однако на развитие архитектуры самих суперкомпьютеров значительное влияние оказали численные методы, которые в свою очередь опираются на достижения и методы фундаментальной математики. Так конвейерный принцип работы центральных процессоров во многом отражает рациональный характер обработки векторов, матриц, полиномов, многопроцессорность вычислительных систем неразрывно связана с методами распараллеливания вычислений и т.д.

Есть ли в настоящее время тенденции к изменению характера численных методов, прежде всего в области задач, ориентированных на применение суперкомпьютеров? Это не праздный вопрос, поскольку в основном превалирует прагматический подход, существо которого можно выразить следующим образом. Увеличение тактовой частоты микропроцессоров, увеличение емкости оперативной памяти, приданной микропроцессору, расширение топологии связей и увеличение частоты передачи данных между процессорами универсально хороши для практически всех численных методов.

Рассмотрим несколько положений, которые можно отнести к категории вероятных тенденций.

1.Выбор и генерация на компьютере среды для вычислений, носящей геометрико-топологический характер и во многом определяющей эффективность самих вычислений на многопроцессорных системах, является часто сложной комбинаторной задачей, требующей значительных вычислительных ресурсов. (Вопросы выбора и склейки сеток из многих миллионов и миллиардов узлов).

2.Необходимость оперативной визуализации трехмерных и более высоких размерностей сложных объектов и явлений для анализа (часто в режиме on-line) и выбора оптимального продвижения вариантов при разработке сложных машин и систем.

3.Проникновение случайных процессов практически во все сферы расчетов, связанных с большими системами. Отсюда Марковские процессы, Марковские поля, случайные распределения и значительный рост комбинаторных вычислений.

4.Разработка математического аппарата, удобного и для описания квантовых систем и для компьютерных расчетов.

Здесь кратко можно говорить о тесно связанной триаде: топология, многомерность, комбинаторика.

Предлагаемая статья рассматривает нетрадиционный подход к кодированию и операциям над кубическими структурами, который позволяет объединить названную выше триаду алгебраической структурой – полугруппой с единицей (моноидом).

На основе установленной в [16] биекции между множеством всех n-разрядных троичных кодов и множеством всех k-граней n-мерного куба в евклидовом пространстве $\mathbf{R}^n$ (подпространство целых точек $\mathbf{Z}^n$), вводится понятие кубанта (кубического кванта)- кода, несущего полную информацию о k-грани в n-кубе. На кубантах задается операция умножения. Расширение троичного алфавита{0,1,2} до четверичного {Ø,0,1,2} приводит к расширению понятия кубанта. На этом расширенном множестве элементов относительно введенной операции рассматривается алгебраическая структура-полугруппа с единицей (моноид). Показано сведение ряда алгоритмов для анализа структуры комплексов из кубантов и вычисления хаусдорфовой метрики на них к алгебраическим операциям над четверичными кодами. Обсуждаются перспективы компьютерной реализации .

Ключевые слова: n-куб, троичное кодирование k-граней, моноид кубантов, хаусдорфова метрика, гамильтоновы циклы, ресурсы суперкомпьютера.

Введение

В основе конструктивного подхода к изучению многих явлений самой различной природы рассматриваются геометрико-топологические модели [1-8], на которые опираются и в которые погружаются численные методы решения задач, которые во многом носят комбинаторный характер. Особое место среди таких моделей занимают решеточные модели, обладающие во многих случаях свойствами, инвариантными по отношению к размерности пространства, а среди них - решетка целых точек (точек с целочисленными координатами) $\mathbf{Z}^n$ в евклидовом пространстве $\mathbf{R}^n$ с заданным ортонормированным репером $\mathbf{e}_1,\mathbf{e}_2,..\mathbf{e}_n$. Единичный n-мерный куб $\mathbf{I}^n$ в $\mathbf{R}^n$ обладает традиционным набором k-мерных граней ($0\leq k\leq n$), при котором грани 0 размерности – вершины (целые точки), грани размерности 1- ребра, грани 2-размерности-квадраты, грани 3-размерности-трехмерные кубы и т.д. до грани n-размерности-самого n-мерного единичного куба. Если

распространить конструкцию $\mathbf{I}^n$ для всех целых точек $\mathbf{R}^n$, то получим бесконечный кубический комплекс, который следуя [9], будем обозначать $\mathbf{R}^n_c$. Широкие исследования кубических комплексов были инициированы С.П.Новиковым [1] в 80-ые годы прошлого века. Этой тематике, в том числе и отображениям на кубические структуры посвящены работы Деза, Штанько, Долбилина, Штогрина [9-11]. Они заложили основу отношения к кубическим комплексам как к объектам, обладающим богатыми возможностями для эффективного отображения многих комбинаторных структур.

С другой стороны в последние годы, развивая направление исследований, связанных с кольцом граней Стенли-Райснера, В.М.Бухштабер и его сотрудники открыли перспективу связи между простыми комбинаторными многогранниками (классическим примером которых является n-куб) и решением дифференциальных уравнений в частных производных Хопфа, Бюргерса и Кортевега- де Фриза.[12,13]

В области актуальных комбинаторных проблем топологии (перечисление, классификация, нумерация) развивается подход, связанный с кубическими диаграммами. [14,15]

Целью предлагаемой статьи является рассмотрение нетрадиционных методов компьютерного представления (кодирования) и операций на кубических комплексах в $\mathbf{R}^n_c$, которые могли бы оказать влияние на архитектуру и состав операций будущих суперкомпьютеров, поскольку именно с суперкомпьютерными системами связаны надежды на решение многих комбинаторных задач.

Кубанты и их свойства

На основе анализа рекурсивной процедуры вычисления коэффициентов пирамиды Паскаля в [16], установлено свойство биекции между множеством всех n-разрядных троичных кодов и множеством всех k-граней n-мерного единичного куба. Это позволяет ввести множество следующих конструктивных элементов, условно называемых **кубантами** (**куб**ический кв**ант**). Такие элементы отражают с одной стороны геометрико-топологические свойства k-мерных граней (кубов) в структуре n-мерного куба и с другой стороны как n-разрядное слово с разрядами из алфавита {Ø,0,1,2} являются элементами алгебраической структуры-моноида (полугруппы с единицей) относительно специально введенной операции умножения, которая будет описана ниже.

Итак пусть в $\mathbf{R}^n$ задан ортонормированный базис $\mathbf{e_1,e_2,\ldots e_n}$ и пусть задано множество всех n-разрядных троичных слов {D} с разрядами d_i из алфавита{0,1,2}.Число таких слов равно 3^n. Между номерами реперных векторов и номерами разрядов в словах установлено взаимно однозначное соответствие $e_i \rightarrow d_i$. Пусть в слове $D=d_1,d_2,\ldots d_n$, часть разрядов принимают значения 2, часть разрядов 1 и

остальная часть 0. Разрядам $d_{i1},d_{i2},\ldots d_{ik}$, принимающим значение 2 ставится в соответствие декартово произведение единичных отрезков, коллинеарных реперным векторам $\mathbf{e}_{i1},\mathbf{e}_{i2},\ldots\mathbf{e}_{ik}$, т.е. k-грань в n-кубе. Остальные разряды со значениями 0 и 1 соответствуют отсутствию или наличию параллельного переноса (трансляции) по направлению реперного вектора, чей номер совпадает с номером данного разряда. Грубо говоря, троичный код отражает два действия-формирование k-грани около (0,0,...0) (разряды со значением 2) и трансляцию этой грани на «свое» место в n-кубе (разряды со значениями 0 и 1). Формально поэтому можно записать для k-грани F:

$$F(k,p)=\Pi\,\mathbf{e}_i \;+\; T\;\mathbf{e}_j;$$
$$i{:}d_i{=}2; \qquad j{:}d_j{=}0,1$$

где k-размерность грани, а p-вершина трансляции.

Отметим, что в принятой кодировке 0-грани (вершины n-куба) совпадают с общепринятыми координатами вершин единичного n-куба.

На рис.1а) показана кодировка всех граней 3-куба (все кубанты в 3-кубе), а на рис.1б) показан кубант 022211 (трехмерная грань в 6-кубе).

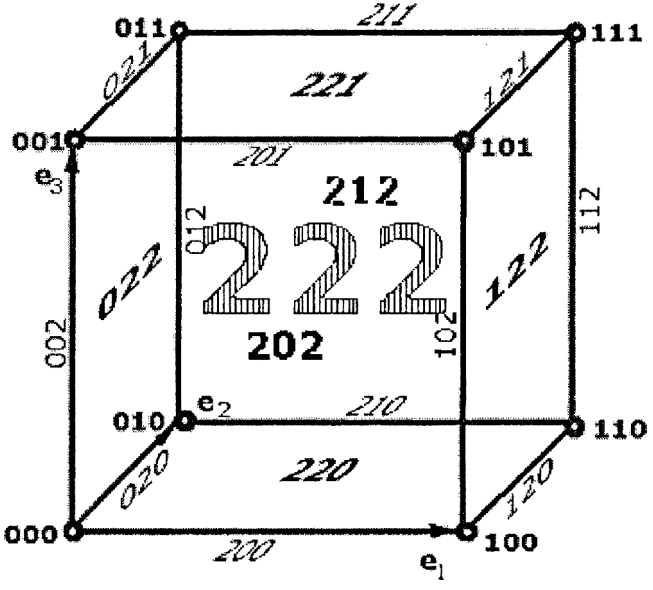


Рис.1 а).Все кубанты 3-куба, 222- весь 3-куб.

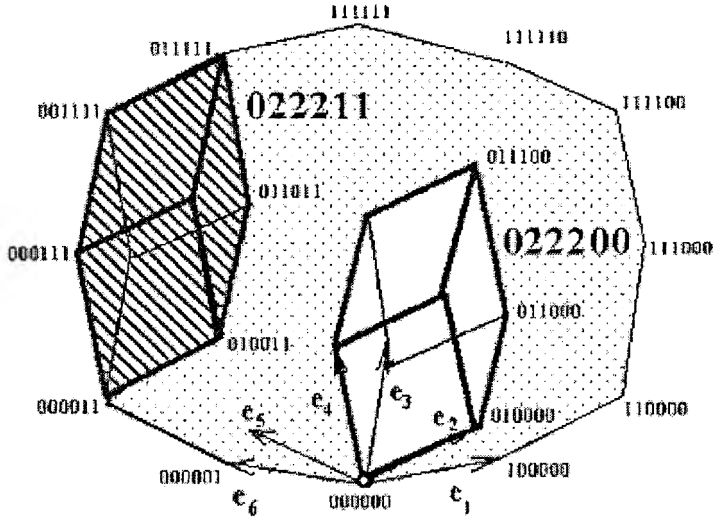


Рис.1 б).Кубант 022211, как трехмерная грань в 6-кубе 022200, транслированная на 000011 (вдоль e5 и e6).

Два кубанта в n-кубе могут находиться в следующих положениях друг относительно друга:

1. Не пересекаться –не иметь ни одной общей точки.(пересечение-пустое множество).
2. Пересекаться- иметь одну общую точку (0-грань-тоже кубант), общее ребро (1-грань), общую 2-грань и т.д., т.е. пересекаться только по кубантам. В частности один кубант может быть целиком частью другого кубанта.

Для определения (вычисления) пересечения кубантов вводится операция умножения, которая является поразрядной и задается следующей таблицей. Здесь алфавит {0,1,2} дополняется символом пустого множества Ø. И таким образом общий алфавит - {Ø,0,1,2}.

x | 0	1	2
0 | 0	Ø	0
1 | Ø	1	1
2 | 0	1	2

Следовательно, появление в произведении двух кубантов по крайней мере в одном из разрядов Ø означает, что эти кубанты не имеют пересечения. Отсутствие в разрядах произведения Ø показывает, что кубанты образуют связное множество и сам **результат умножения есть общий кубант** (общая грань). В дальнейшем будет использована для обозначения произведения кубантов префиксная форма. Так произведение кубантов D_1=220200 и D_2=220211 будет записано: $\Pi(D_1,D_2)$= 2202ØØ (пересечение отсутствует).

В расширенном четверичном алфавите слова соответствуют реальным k-граням без разрядов со значением Ø (кубанты) и не соответствуют граням в случае наличия в их разрядах Ø (псевдокубанты). Однако будем рассматривать их часто совместно , поскольку они вместе образуют **полугруппу с единицей - моноид.** Как нетрудно видеть такой единицей в моноиде является кубант 22...2.

Отметим, что длина минимального пути (по ребрам, т.е. по 1-граням, имеющим длину равную 1) между двумя непересекающимися кубантами в n-кубе равна числу разрядов со значением Ø в произведении этих кубантов. Будем обозначать эту величину через ω. Так в выше приведенном примере $\omega(\Pi(D_1,D_2))=2$ и $Lmin(D_1,D_2)=2$. В случае вершин n-куба длина минимального пути совпадает с хэмминговым расстоянием между соответствующими двоичными кодами. Вычисление последовательности ребер, по которым реализуется путь минимальной длины, более наглядно представить на следующем примере. Пусть заданы D_1=220210 и D_2=021121.Расположим их один под другим таким образом, чтобы между ними можно было поместить два кода (по значению для этого примера $\omega(\Pi(D_1,D_2))=2$). Затем последовательно для каждого случая, когда по вертикали комбинация в разряде (0,1) или (1,0) производим вставление 2, как это показано ниже:

```
      220210  220210  220210  2 2 0 2 1 0
                                          Проход по ребру 002110
D*    ---------→ 02?11?→022110  0 2(2)1 1 0
                                          Проход по ребру 001112
D**   ---------  02?11?  02111?→0 2(1)1 1(2)
                                          П(021112,021121)≠Ø;
      021121  021121  021121  0 2 1 1 2 1
```

Множество кубантов можно рассматривать как кубический комплекс в n-кубе, свойства которого (связность, структура дуального комплекса, отношения с другими комплексами) **вычисляются алгебраически** прежде всего на основании свойств моноида.

Кубический комплекс в n-кубе в общем случае может и не обладать свойством выпуклости, поэтому наиболее естественна **хаусдорфова метрика**, которая является обобщением хэмминговой метрики между двоичными кодами. Хаусдорфова метрика (H-метрика) между двумя кубантами определяется как

$$\rho_H(D_1,D_2)=Max\{Lmin(D_1,D_2*/D_1),Lmin(D_2,D_1*/D_2)\},$$

где D_1*/D_2 и D_2*/D_1 - «сжатия» кубантов по отношению соответственно к D_2 и D_1, определяемые в соответствии с поразрядными действиями, задаваемые ниже приведенными таблицами.

D_1	0	1	2
D_2 |			
0	0	1	1
$D_1*/D_2 \rightarrow$ 1	0	1	0
2	0	1	2

D_2	0	1	2
D_1 |			
0	0	1	1
$D_2*/D_1 \rightarrow$ 1	0	1	0
2	0	1	2

Грубо говоря такое сжатие соответствует выделению в одном кубанте наиболее удаленной части (также кубанта) по отношению к другому.

Так для кубантов 6-куба D_1=022211 и D_2=112222 соответствующие D_1*/D_2=002211 и D_2*/D_1=112200. Отсюда:

$$Lmin(D_1,D_2*/D_1)=\omega(\Pi(D_1,D_2*/D_1))=\omega(Ø122ØØ)=3;$$

$$Lmin(D_2,D_1*/D_2)=\omega(\Pi(D_2,D_1*/D_2))=\omega(ØØ2211)=2;$$

$$p_H(D_1,D_2)=\max\{3;2\}=3; \quad (1)$$

На рис.2а) эскизно (с убиранием деталей) показана эта ситуация в 6-кубе. На рис.2б) показана ситуация для случая двух комплексов $K1=\{D_1,D_2,D_3,D_4,D_5,D_6\}$; и $K2=D_7$,где D_1=220000;D_2=112200;D_3=111122;D_4=000022;D_5=002211;D_6=221111;D_7= 122021; Применяя алгебраически вышеизложенные методы, легко установить циклическую структуру двумерного комплекса K1 и Lmin(K1,K2)=1.

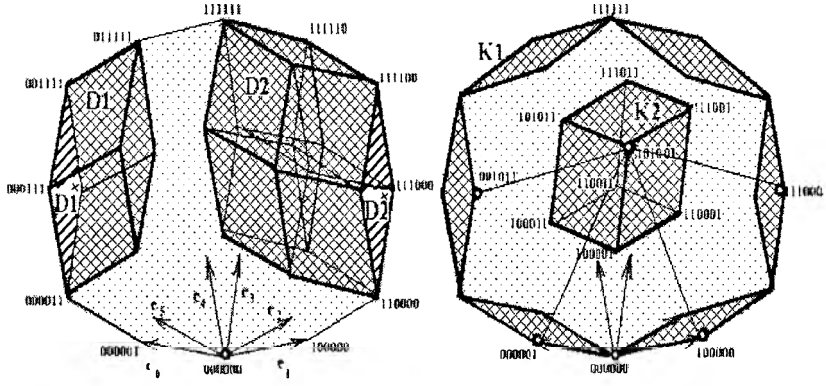


Рис 2. а).К вычислению хаусдорфовой метрики между кубантами D1 и D2 в 6-кубе, б).Взаимное положение комплексов K1 и K2 в 6-кубе.

Итак, кубанты n-куба образуют **метрическое пространство с хаусдорфовой-хэмминговой метрикой.** В [9] такие пространства отнесены к так называемым гиперметрическим пространствам. Поэтому более корректно (1) обозначить как

$$p_{HH}(D_1,D_2)=3;$$

Поскольку кубанты (кроме размерности 0) являются множествами вещественных точек, то для них метрическое пространство может быть определено и с евклидовой-хаусдорфовой метрикой, для которой:

$$p_{EH}(D_1,D_2)= p_{HH}(D_1,D_2)^{1/2};$$

Это существенно, когда хаусдорфова метрика рассматривается на комплексах из кубантов.

Рекурсивная процедура построения гамильтонова цикла в n-кубе

Представим гамильтонов цикл в n-кубе как последовательность ребер, инцидентных двум следующим друг за другом вершин в цикле. Поскольку ребра являются одномерными кубантами, будем формулировать свойства такого цикла ребер в терминах кубантов и псевдокубантов, т.е. используя алфавит {Ø,0,1,2} и свойства моноида, образованного всеми n-разрядными кодами в этом алфавите относительно опреации умножения [ ]

Рассмотрим в такой последовательности из n кубантов (ребер) любые три последовательных кубанта D_{i-1},D_i,D_{i+1} ($1<i<n$; $D_{n+1}=D_1$) .Тогда из определения гамильтонова цикла:

1). $D_{i-1}\neq D_i\neq D_{i+1}$;

2). $\omega(\Pi(D_{i-1},D_i))=0$, $\omega(\Pi(D_i,D_{i+1}))=0$, $\omega(\Pi(D_{i-1},D_{i+1}))=1$;

3). $w(\Pi(D_i,D_k))\neq 0$; для всех i и $k\neq i-1,i+1$;

В дальнейшем будем обозначать некоторый гамильтонов цикл в виде последовательности кубантов-ребер в n-кубе через $H(n)=D_1,D_2,D_3,\ldots D_k$, где $k=2n$ и D_i n-разрядный код со значениями разрядов из {0,1,2}, при этом лишь один из разрядов имеет значение 2 (номер этого разряда равен номеру реперного вектора, которому коллинеарно данное ребро).

Рассмотрим следующую процедуру, состоящую из шагов.

1. Образуем последовательность $H^*(n)=D_k,D_{k-1},D_{k-2},\ldots D_1$.

2. Образуем $H(n/0)=D_{10},D_{20},\ldots D_{k0}$, где $D_{i0}=D_i+0_{n+1}=(d_{i1},\ldots d_{in},0)$, и

$H^*(n/1)=D_{k1},D_{k-1,1},D_{k-2,1},\ldots D_{11}$, где $D_{j1}=D_j+1_{n+1}=(d_{j1},\ldots d_{in},1)$.

3. Вычисляем B_1 и B_2 такие, что $\Pi(D_{10},B_1)=D_{11}$ и $\Pi(D_{10},B_2)=D_{11}$. Поскольку D_{10} и D_{11} отличны друг от друга лишь в одном n-ом разряде и имеют один и тот же номер разряда, имеющий единственно значение 2 (пусть этот номер равен m, при этом m=n), то $B_1=(d_{11},d_{12},\ldots d_{1m}=0,d_{1m+1},\ldots d_{1n-1},2)$ и $B_2=(d_{11},d_{12},\ldots d_{1m}=1,d_{1m+1},\ldots d_{1n-1},2)$.

4. В $H(n/0)$ удаляем первый кубант D_{10} и добавляем кубант B_2 после D_{k0} и аналогично для $H^*(n/1)$ удаляем D_{11} и на его

место добавляем B_1. Затем склеиваем эти две последовательности в единую и получаем:

$$D_{20},D_{30},\ldots D_{k0},B_2,D_{k1},D_{k-1},1,\ldots D_{21},B_1=H(n+1);$$

Для того, чтобы показать, что это гамильтонов цикл, достаточно только рассмотреть следующие последовательные тройки; $,D_{k0},B_1,D_{k1}$; и D_{21},B_2,D_{20}; но они по построению удовлетворяют условиям (1). Для полного определения рекурсивности предложенной процедуры достаточно сделать начальный шаг в рекурсии, т.е. продемонстрировать построение H(3) из H(2). Для 2-куба (квадрата):

H(2) =02,20,12,21;
H*(2)=21,12,20,02;
H(2/0) =020,200,120,210;
H*(2/1)=211,121,201,021;
B1=002; B2=012;

И отсюда: H(3)= 200,120,210,012,211,121,201,002 ; (рис.3)

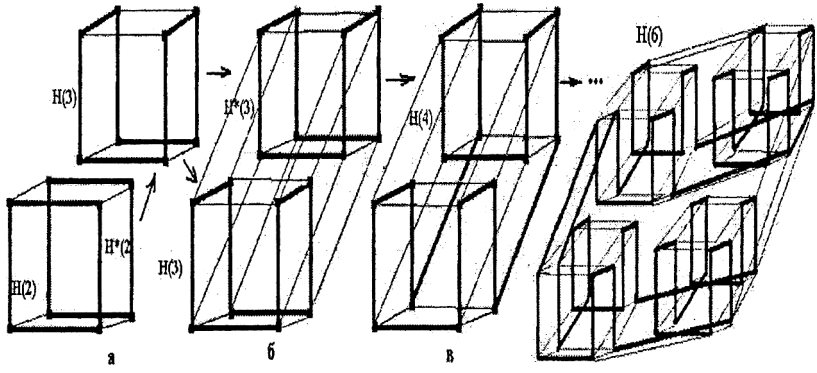


Рис.3. Ребра, входящие в гамильтоновы циклы (H(2),H(3),H(4),H(6)) показаны толстыми линиями.

Основная идея процедуры состоит в построении в $\mathbf{I}^{n+1}$ двух несвязных между собой циклов, которые образуются из гамильтонова цикла в $\mathbf{I}^n$ и принадлежат непересекающимся гиперграням в $\mathbf{I}^{n+1}$ (за счет приписывания в последний n+1 –ый разряд кубантов 0 и 1). У этих циклов ровно $k=2^n$ пар кубантов-ребер, каждую такую пару можно вырезать и заменить их парой, объединяющих оба цикла в один, не

нарушая основного положения в определении об одинарном прохождения цикла через каждую вершину I^{n+1}.

Отметим, что для двух ортогональных (не совпадающих ни по одному ребру) гамильтоновых циклов H_1 и H_2 (пример для I^4 показан на рис.4) расстояние в евклидово-хаусдорфовой метрике $\rho_{EH}(H1,H2)=1/2$;

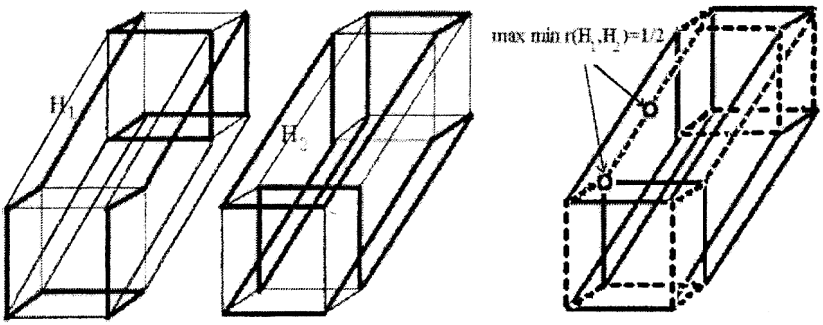


Рис.4. Слева ребра двух ортогональных гамильтоновых циклов в I4 показаны толстыми линиями. Справа они совмещены для наглядности, что max min ρEH(H1,H2)=1/2.

Кубические комплексы в R^n_c

В данной статье мы остановимся только на одном, наиболее прикладном аспекте, подчеркивающем конструктивный характер предложенных методов кодирования. Но прежде вернемся к n-кубу и рассмотрим множество всех подмножеств (булеан) его кубантов, т.е. множество всех комплексов n-куба. Число элементов этого множества равно 2^A, где $A=3^n$. Кубант при выборе «машинного» алфавита {0,1,2,3}при замене (Ø→0;0→1;1→2;2→3),можно рассматривать как число, записанное в четверичном виде. Поэтому каждую строку из кубантов, соответствующую некоторому комплексу, можно выстроить в лексикографическом порядке, т.е. на первом месте в строке кубант с наименьшим числом и затем далее в порядке возрастания. С другой стороны каждому комплексу соответствует однозначно двоичный номер в булеане и значит все комплексы n-куба перенумерованы и каждый, как строка слов упорядочен.

Из этого общего построения мы воспользуемся частным случаем. При рассмотрении трехмерного случая ограничимся только двумерными кубантами, т.е. 2-гранями или просто квадратными гранями, число которых в 3-кубе равно 6. Тогда общее число комплексов только из таких граней (кстати гиперграней для 3-куба) равно $2^6=64$ и весь булеан (множество всех подмножеств) для этого

случая представлен ниже вместе с нумерацией, а типы комплексов представлены на рис.5, где показано построение кубической бутылки Клейна из 2-комплексов, как из готовых панелей.

Комплексы, их кубанты и номера комплексов:

Ø-0;
1 кубант (022)-1;(122)-2;(202)-3;(212)-4;(220)-5;(221)-6;
2 кубанта (022,122)-7;(022,202)-8;...(220,221)-21;
3 кубанта (022,122,202)-22;(022,122,212)-23;...(212,220,221)-41;
4 кубанта (022,122,202,212)-42;(022,122,202,221)-43;...
(202,212,220,221)-56;
5 кубантов (022,122,202,212,220)-57;(022,122,202,212,221)-58;
...(122,202,212,220,221)-62;
6 кубантов (022,122,202,212,220,221)-63;

Собственно происходит отображение номеров соответствующих комплексов-панелей в трехмерный индексный массив памяти компьютера, где под каждый кубик 1x1x1 в данном случае достаточно одного байта (всего 64 различных номеров комплексов с учетом вращений комплексов вокруг осей симметрии 3-куба). Поскольку массив 5x3x5=75 байтам, можно говорить о бутылке Клейна «емкостью в 75 байт». Хотя это отнюдь не минимальная по объему памяти бутылка. В приведен более компактный вариант кубической бутылки Клейна.

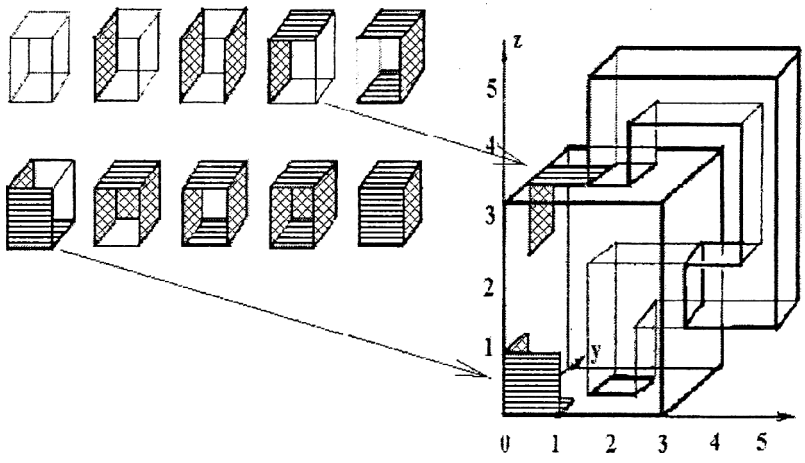


Рис.5. Кубическая бутылка Клейна из 2-комплексов-панелей емкостью 75 байт

Этот метод топологического панельного конструирования, показанный на этюдном примере, может быть перенесен на более высокие размерности.

Обсуждение результатов

Прежде всего, отметим полиморфизм введенного кубанта, поскольку его можно рассматривать как:

1.Число в троичном или четверичном виде.

2.Геометрический объект- k-мерная грань в n-мерном кубе.

3.Топологический объект-элемент кубических комплексов.

Наиболее значимым представляется возможность представления **многомерных кубических комплексов в троичном алфавите**, а с добавлением пустого множества в четверичном алфавите для образования моноида (алгебраической структуры) с введенной операцией умножения кубантов.

Этот метод кодирования и единая операция над словами, совмещающими в своих символах различные математические свойства, представляется как пример **максимального (поразрядного) параллелизма** в кодировании.

Рассмотрим некоторые вычислительные аспекты предложенных методов.

1.Релизация поразрядного умножения кубантов потенциально обладает неограниченным параллелизмом (точнее совмещением) при реализации на компьютере, поскольку ее можно рассматривать как логическое перемножение сколь угодно длинных четверичных (два бита) стрингов, которое может выполняться за один такт.

2.Введение хаусдорфова сжатия кубантов вместе с определением минимального пути между кубантами и комплексами переводит вычисление хаусдорфовой метрики из задач сложности 2^n в разряд задач полиномиальной сложности. Это требует некоторого пояснения.

Пусть в $\mathbf{I}^n$ заданы два кубанта D_1 и D_2 размерности n/2. Тогда перечисление всех целых точек каждого кубанта (для последующего вычисления max min $L(D_1,D_2)$) приведет к множествам размерности порядка $2^{n/2}$, в то время когда вычисления с хаусдорфовым сжатием требуют порядка n^2 операций.

3.Перечисление и нумерация комплексов из гиперграней n-мерного куба могут быть оценены с точки зрения затрат памяти компьютера следующим образом.

Для $\mathbf{I}^n$ число гиперграней (размерность n-1) равно $F(k)= C^k_n\, 2^{n-k}$ при k=n-1 , отсюда число различных комплексов на этих гранях равно $2^{F(k)}$ и число разрядов под номера комплексов равно F(k). Для оценки емкости памяти в байтах обозначим через Fb(k)=F(k)/8 и размер решетки для отображения комплексов X^n. Таким образом объем памяти

можно оценить как $X^nFb(k)$ и отсюда при оперативной памяти суперкомпьютера в 10^{15}байт и X=100 возможна работа для размерностей вплоть до 6d.

Заметим, что размерности 10d-11d , представляющие интерес в теорстической физике, при таких допущениях на емкость памяти допускают обработку решеток с X=20-30.

4.Для оценки затрат машинного времени при обработке объектов высокой размерности будем исходить из режима одноразового прохода по решетке 100^6 (для $\mathbf{Z}^6$) и замене в каждом шестимерном кубе номера комплекса в зависимости от содержимого соседних кубов (простейший вариант динамики при ближайшем зацеплении). Число операций такого локального преобразования положим пропорциональным числу соседних кубов-$\lambda 12$. Отсюда перестройка комплексов при одноразовом проходе по всей решетке потребует примерно $\lambda 10^{13}$ операций и при $\lambda<100$ оценка дает 10^{15} операций. Это дает возможность на современных суперкомпьютерах допускать многоразовые проходы на таких решетках для выявления тех или иных асимптотических тенденций перестроения комплексов. А в ряде случаев переходить к марковским моделям для установления эргодических свойств .[17].

Предположение столь простой линейной зависимости ($\lambda<100$) сугубо предварительно только для общей ориентации с ресурсами суперкомпьютера.

Отметим, что в развитие предложенного подхода естественно рассмотреть:

1.Возможность близкого подхода к кодированию симплициальных комплексов.

2.Действия симметрической группы подстановок (и ее подгрупп) на кубанты.

3.Случайное троичное n-разрядное слово, как случайный кубант в n-кубе и множество таких слов как случайный кубический комплекс с привлечением аппарата марковских цепей к анализу эргодических свойств случайных комплексов.

Предложенные методы развиваются в рамках инструментальной системы «Топологический процессор»[18], ориентированной на реализацию на суперкомпьютерах кластерного типа, в частности на суперкомпьютер МГУ «Чебышев» и суперкомпьютеры петафлопсного уровня.

В заключение отметим, что в развитии суперкомпьютерного направления вместе с прогрессом в традиционных ресурсах - объеме памяти, числе процессоров, топологии их соединения, повышения тактовой частоты и т.д., заметную роль может сыграть математическое обеспечение и его аппаратная поддержка в области анализа и генерации геометрико-топологической среды вычислений. Здесь достижения

алгебраической геометрии и топологии, теории категорий, трансформированные в новые формы кодирования и состав нетрадиционных операций, предварительно отработанных на инструментальных системах, могут существенно повысить реальную производительность суперкомпьютеров.

Автор выражает благодарность за внимание и поддержку работы В.А.Садовничему, Л.Н.Королеву, А.В.Тихонравову, А.Н.Томилину. Работа поддержана грантом РФФИ (09-07-12135-офи_м).

Литература

1. С.П.Новиков.Топология.Москва-Ижевск:РХД,2002
2. Yu.I.Manin. Classical computing, quantum computing and Shor,s factoring algorithm.// arXiv:quant-ph/9903008v1 2Mar 1999.
3. V.Kaibel, G.Ziegler. Counting Lattice Triangulations.// arXiv:math/0211268v2[math.CO] 13 Dec 2002.
4. F.Lutz. Triangulated Manifolds with Few Vertices: Geometric 3-Manifolds.// arXiv:math/0311116v1[math GT] 7Nov 2003.
5. P.Collet, J.Eckmann. Dynamics of Triangulations.// arXiv:math-ph/0412085v1 23Dec 2004
6. F.Ardila, R.Stanley. Tilings.// arXiv: math/0501170v3[math.CO] 25Jan 2005
7. V.Desoutter,N.Destainville. Flip dynamics in three dimensional random tilings.// arXiv: cond-mat/0406728v3 [cond-mat.stat-mech] 8Nov 2004
8. J.Ambjorn, J.Jurkiewicz, R.Loll. The Universe from Scratch.// arXiv: hep-th/0509010v3 14Oct 2006.
9. Н.П.Долбилин, М.А.Штанько, М.И.Штогрин. Кубические многообразия в решетках//Изв.РАН, сер. матем.1994.Т.58, вып.2.С.93-107
10. М.Деза, М.Штогрин. Вложение графов в гиперкубы и кубические решетки.//УМН.1997. т.52 №6.С.155-156.
11. М.Деза, М.Штогрин. Мозаики и их изометрические вложения.// Изв.РАН. Сер.матем.2002.т.66№3.С.3-22.
12. В.М.Бухштабер,Т.С.Панов. Торические действия в топологии и комбинаторике. //МЦНМО.2004
13. В.М.Бухштабер. Кольцо простых многогранников и дифференциальные уравнения.//Тр.МИАН.2008,263,18-43
14. S.Baldridge, A.Lowrance. Cube diagrams and a homology theory for knots.// arXiv:0811.0225v1 [math.GT] 3 Nov2008
15. S.Baldridge, B.McCarty. Small examples of cube diagrams of knots.// arXiv:0907.5401v1 [math GT] 30Jul2009
16. Г.Рябов.О путевом кодировании k-граней в n-кубе.//Вычислительные методы и программирование.2008.т.9№1.С.16-18

17. Г.Рябов. Марковские цепи в динамике примитивной триангуляции в пространствах $\mathbf{R}^3$ и $\mathbf{R}^4$.//Вычислительные методы и программирование.2009.т.10№1.С.1-8.
18. Ryabov G.,Serov V. Simplicial-lattice model and metric-topological constructions.// Proc. of the IX Conf. on Pattern Recognition and Information Processing. Minsk. 2007. Vol 2.135-140.

Королев Л.Н.

Оценка вычислительной сложности некоторых алгоритмов построения классификаторов

Работа выполнена при поддержке гранта РФФИ 08-07-00445

Цель статьи – предложить методику оценки глобальной вычислительной сложности и временные затраты некоторых алгоритмов прямого перебора, возникающих при решении задач построения классификаторов, кластеров и решателей на компьютерах с массовым параллелизмом. Перебору подлежат параметризованные критерии близости с использованием нейросетей и принципов «генетического программирования». Полученные оценки позволят прикинуть размерности векторных пространств признаков классифицируемых объектов, с которыми справятся современные вычислительные системы.

В связи с появлением вычислительных систем с терафлопной и петафлопной производительностью, имеются в ввиду такие системы как BladeCenter QS22/LS21, Cray XT5, BG/S, Cray BlackWidow, Cray Baker, а также BG/P и кластер Скиф –«Чебышов», установленные в Московском Государственном Университете, стало возможным решать многие практически важные задачи методами прямого перебора, доставляющими точное решение. К этой категории задач принадлежат некоторые оптимизационные задачи поиска глобального экстремума. Решение таких обычно плохо формализованных задач, для которых отсутствуют математические модели, точными классическими приемами практически невозможно. Эвристические методы, сокращающий полный перебор применяемые в таких случаях, всегда оставляет сомнение в получении «глобально правильного» ответа на вопрос, поставленный в исходной задаче.

В данной статье мы будем рассматривать переборные алгоритмы, требующие сравнительно малой доли вычислений по сравнению с обращениями к памяти. К таким задачам сводится конструирование анализаторов ДНК, ряда задач фармакологии, биоинформатики и т. д.

В качестве примера мы рассмотрим задачу автоматического построения классификаторов в условиях, когда нам заданы только выборки из классов, и перечень самых общих критериев близости. Применение прямого перебора для построения алгоритмов классификации позволит оценить размеры вектора признакового пространства, тех его размеров, с которыми могут справиться современные вычислители.

Всякая интеллектуальная деятельность в конечном итоге связана с построением алгоритмов анализа пространства признаков, описывающих тот или иной объект, ту или иную ситуацию. Очевидно, чем точнее и полнее описаны объекты и ситуации, тем более правильными окажутся рассуждения по их анализу и принятию решений. Размер вектора признаков в каком то смысле является мерилом наших знаний о предмете, а способность их обрабатывать – мерилом интеллекта вычислительных систем.

Попытка определить потенциально допустимые размеры вектора признаков (арности пространства признаков), с которые могут справиться современные суперЭВМ, представляется заманчивой. Может быть, неудержимый процесс гонки за повышением производительности вычислительных систем связан с желанием обрабатывать объекты, выраженные сотнями тысяч признаков.

По-видимому, невозможно построить общую теорию определения максимально допустимой размерности вектора признаков, доступной для обработки на конкретной ЭВМ с заданной производительностью. Но для конкретного алгоритма обработки это можно сделать.

Направление прикладных исследований, связанных с построением решателей оптимизационных задач, в настоящее время интенсивно развивается в направлении использования систем *обучения* вычислительных машин ("*Machine Learning - ML*").

Это связано с решением важных плохо формализуемых прикладных задач прогнозирования, распознавания образов, задач оптимизации, не поддающихся иным методам решения.

К направлению *ML* традиционно относят эволюционные и генетические алгоритмы, генетическое программирование, нейросетевые вычисления (нейрокомпьютинг), клеточные автоматы. Все эти алгоритмы в той или иной степени являются переборными по своей сути. Именно подобного рода методы и алгоритмы решения задачи конструирования классификаторов будут рассматриваться в данной статье в расчете на возможности использования машин с массовым параллелизмом.

Общим для большинства задач *ML* является то, что в качестве исходных данных задаются или автоматически формируются обучающие выборки, манипулируя с которыми стараются получить алгоритмы (функции), решающие поставленные задачи. В большинстве своем методы основаны на эвристиках и на вычислительных экспериментах, подтверждающих адекватность полученных решений.

Обучающие выборки, как правило, составляют очень небольшую часть элементов исследуемого множества, над которыми строятся алгоритмы, доставляющие решение задачи. В задачах конструирования функций, разделяющих множества на классы,

исходные обучающие выборки могут составлять тысячные и многомиллионные доли процента от объема классов, и их объем как правило недостаточен для получения точного алгоритма классификации, т. е. алгоритма, гарантирующего правильность построенного классификатора.

Тем не менее, использование принципов и методов *ML* позволяет, даже в условиях ограниченности размеров выборок, получать приемлемые результаты.

Прямой перебор, как правило, просто распараллеливается и в ряде случаев позволяет избежать вычислительную и логическую сложности, возникающие при построении оптимальных алгоритмов поиска решений.

Цель статьи – попытаться оценить размеры вектора признаков, допустимые для обработки на ЭВМ с заданной пиковой производительностью, с учетом ограничений на допустимое время решения, в привязке к некоторым простым переборным алгоритмам методами ML на современных массивно параллельных вычислительных системах.

Мы будем в этой статье в основном рассматривать задачу обучения (настройки) параметризованных алгоритмов построения классификаторв с использованием моделей искусственных нейронных сетей и генетических алгоритмов.

Методы генетического программирования для конструирования оптимальных алгоритмов, формул, электронных схем и компьютерных программ, используются достаточно давно [1].

Основная идея применения генетического программирования для конструирования программ-классификаторов состоит в следующем. Процесс обучения машин, с целью создания программ-классификаторов, можно рассматривать как итерационный процесс, генерирующий программы, удовлетворяющие ряду условий, и выбора из них наиболее приемлемых для классификации (наиболее лучших по некоторым критериям).

Для этого необходимо определить общую структуру генерируемых компьютерных программ, из которых будут формироваться популяции в шагах итерационного процесса генетического программирования, и определить состав операций построения новых объектов.

Для оценки пригодности сгенерированных программ-классификаторов используются обучающие выборки. Если удается, используя только выборки, построить классификатор, то это будет означать, что в выборке в том или ином виде содержится информация, достаточная для расшифровки алгоритма классификации.

Классификатор может быть представлен в виде программы, составленной на некотором специфическом языке, содержащем свой

перечень операторов, переменных и отношений между ними. Эксперт или заменяющая его автоматическая система строит дерево рассуждений (решающее дерево), которое в узлах содержит функции, принимающие конечное число целочисленных значений, указывающих номера ребер, по которым следует продолжать движение по дереву до его листьев – указателей номеров классов принадлежности рассматриваемого элемента.

Любую компьютерную программу можно считать двоичным деревом решений, по той простой причине, что в арсенале системы команд любой традиционной машины условный переход имеет только два выхода. Условие, которое проверяет эта команда, можно считать результатом вычисления некоторой булевой функции от многих переменных, принимающей два значения – 0,1.

Представим себе, что нам удалось сконструировать программу, которая правдоподобным образом умеет классифицировать элементы рассматриваемого множества. Как определить, что она делает это правильно?

Прежде всего, эта программа должна любой элемент, принадлежащий выборкам, оставлять в их классе, а любой другой проверяемый элемент, не принадлежащий выборкам, относить только к одному из классов (в предположении, что классы не пересекаются).

В инструментальной системе, генерирующей программы для итераций генетического алгоритма, должен быть блок, проверяющий эти условия и разделяющий генерируемые программы на допустимые и недопустимые.

Сразу же отметим, что этим простейшим необходимым условиям удовлетворяет целое множество допустимых программ, в частности программы, относящие к классам только элементы самих выборок.

Интуитивно ясно, что классификатор может считаться хорошим, если он относит к некоторому классу только элементы *близкие* по каким-то критериям к выборке из означенного класса. Для того, чтобы автоматически конструировать программы-классификаторы, основанные на понятии близости, отношение близости должно быть строго формализовано.

Алгебраический подход к проблеме классификации и распознавания образов, развиваемый школой Ю. И. Журавлева (аксиомы в алгебраическом подходе) [3]), решает многие теоретические проблемы в этой области. Разработку системы аксиом в этой теории можно интерпретировать, как некоторый способ формализации отношения близости.

Пусть мы имеем в распоряжении несколько различных формальных (алгоритмизируемых) определений близости. Какое

отношение близости при этом окажется предпочтительнее для системы генерации классификаторов?

Ответ на этот вопрос получается обычным способом, принятым в обучающих системах: выборка делятся на обучающую и тестовую последовательности. На тестовой последовательности проверяется адекватность выбора отношения близости.

Выше приведенные рассуждения диктуют следующую общую простую схему применения генетического программирования к практическому конструированию программ-классификаторов:

- Формирование множества формальных критериев (определений, аксиом) близости.
- Попытка конструирования по каждому из критериев близости наилучшей программы классификатора.
- Выбор «наилучшей» из полученного множества программ-классификаторов, построенных по разным критериям.

В отличие от классических генетических алгоритмов, основанных на вероятностном подходе к формированию популяций программ-классификаторов, мы рассмотрим возможность использования прямого перебора, и постараемся оценить пределы его реализуемости на современных высокопроизводительных вычислительных системах с учетом размерности вектора признаков.

Формирование множества формальных критериев (определений) близости.

Рассмотрим несколько типов критериев близости, используемых в конкретных приложениях, на основе которых можно генерировать последовательности тестирующих функций для использования их в генетическом программировании. При решении практических задач таких критериев не так уж много.

Везде далее мы будем рассматривать множества из элементов, состоящих из двоичных последовательностей одинаковой длины (двоичных векторов или слов).

Имеются следующие два типа критериев близости, которые, по-видимому, охватывают большинство практических применений:

- Логическая близость по набору переменных.
- Близость по структуре, в которую входит близость по критерию встречаемости фрагментов слов, наличию или отсутствию некоторых заданных фрагментов, наличию искомых отношений между фрагментами слова и т. д.

Логическая близость по набору переменных определяется следующим образом:

Два элемента множества $x=(x_1, x_2, \ldots, x_n)$ и $y=(y_1, y_2, \ldots, y_n)$ назовем близкими по набору переменных $P=\{\alpha_1, \alpha_2, \ldots \alpha_k\}$, если

$x_{\alpha_i} = y_{\alpha_i}$ для всех $i=\overline{1,k}$, т. е. когда x и y в позициях с номерами $\alpha_1, \alpha_2, \ldots \alpha_k$ совпадают.

Конкретных критериев *близости по структуре* достаточно много и они зависят от того, что будет пониматься под структурой элемента в каждом конкретном случае. Общего определения структуры, по-видимому, дать невозможно, но в каждой предметной области конкретных критериев близости, как правило, небольшое количество, определяемое принятыми в данной области знаний методами исследований. Примером эксплуатации критерия близости по структуре являются алгоритмы множественного и парного выравнивания при изучении генетического кода [4, 5], близость по числу различий в заданных позициях сравниваемых слов (расстояние по Хеммингу), близость по подобию рельефов признаковых слов и т. д.

Отношение близости $\eta(x,y)$ между двумя элементами множества, которым мы будем пользоваться, определим как функцию, принимающую два значения (0 или 1) и обладающую свойством симметричности и транзитивности:

$$\eta(x,y) = \eta(y, x)$$

$$\eta(x,y) \wedge \eta(x,z) = \eta(z,y)$$

(Если $\eta(x,y) = 1$ и $\eta(x,z) = 1$, то $\eta(z,y) = 1$).

Если дан конечный набор элементов (выборка) и для каждого элемента этого набора построено подмножество элементов, находятся в отношении близости, то объединение всех подмножеств, близких для всех элементов выборки, мы назовем *кластером* близких к этой выборке элементов .

Кластеризация разделяет всё множество объектов на подмножества, которые необязательно соответствуют реальным классам исходной задачи. Однако, гипотеза, утверждающая, что кластеры близких элементов входят в состав соответствующего класса, имеет право на существование, если критерий близости определен верно.

Классификация сводится к тестированию. Теория тестирования в применении к задачам типа прогнозирования и распознавания образов разработана давно научными школами В. С. Яблонского и Ю. И. Журавлева и успешно применяется при решении многих важных практических задач.

Оценка переборного алгоритма на основе критерия логической близости по набору переменных

Мы будем рассматривать задачу классификации в обычной постановке: задано множество $V = \{ v_1, v_2, \ldots v_N\}$, каждый элемент которого $v_i = (x_{1i}, x_{2i}, \ldots x_{ni})$, представляет набор (вектор) двоичных

признаков x_{ji}, характеризующих i-ый элемент множества. Пусть о множестве V известно, что оно содержит конечное число подмножеств $K_1, K_2, \ldots, K_m$, называсмых классами. Заданы также конечные выборки элементов, принадлежащих этим кассам: $E_1 \subset K_1 \subset V$, $E_2 \subset K_2 \subset V$, ..., $E_m \subset K_m \subset V$.

В терминах тестовых таблиц, предложенных еще в 1955 г. В. С. Яблонским, это означает, что $V = \{v_1, v_2, \ldots v_n\}$ представляет собой конечное множество признаков (вопросов), а столбцы таблицы представляют собой классы (функции).

В тестовых таблицах, с которыми реально приходится работать, столбцы классов заполнены частично, т. е. тестовая таблица задана не полностью.

Упорядоченный в лексикографическом порядке список всех слов исходного множества можно мыслить себе как вход в таблицу истинности (ТИ) некоторой булевой функции, принимающей единичные значения в строках, которые представляют элементы, относящиеся к определенному классу. Иными словами, с каждым классом связана соответствующая булева функция и программа, вычисляющая эту булеву функцию.

По аналогии с таблицей истинности (ТИ), задающих булеву функцию, рассмотрим таблицу меток классов (ТМК), которая каждому слову из множества V ставит в соответствие идентификатор класса, к которому это слово принадлежит, например, целое число, обозначающее номер класса (тестовая таблица в определениях Яблонского С. В. [6]). Каковы бы ни были по сложности реальные критерии отнесения объектов к классу, теоретически эти критерии могут быть реализованы соответствующей расстановкой единиц в ТИ. Если исходное множество разделено на несколько непересекающихся классов, то будет индуцировано несколько булевых функций, которые можно объединить в одну таблицу. Поскольку таблицы истинности этих функций не пересекаются по единицам, то можно получить непротиворечивую таблицу меток классов. Таблицу меток классов можно рассматривать как табличное задание некоторой целочисленной функции.

Выборку можно считать кластером близких друг к другу элементов, и свести задачу к построению на этой основе конструктивного алгоритма построения отношения близости и соответствующего кластера элементов, близких к данной выборе по выбранному критерию близости, расширяющего выборку.

Булева функция, проверяющая принадлежность любого элемента множества V к классу, характеризуемому выборкой E_A, в своей таблице истинности должна обязательно содержать единицы в строках, соответствующих элементам этой выборки, и обязана содержать нули в строках таблицы, соответствующих элементам выборок из всех других

классов. На этом простом соображении можно построить переборный алгоритм кластеризации. Знание состава выборок, принадлежащих разным классам, позволяет нам частично заполнить таблицы истинности тестирующих булевых функций, проверяющих принадлежность элементов к классам, из которых взяты выборки.

Исходные данные для алгоритма конструирования кластера, который нет имеет пересечений с выборками из других классов:

- Даны m выборок – E_1, E_2, ... E_m, принадлежащих m различным классам: $K_1, K_2, \ldots K_m$.
- Заданы длины этих выборок – $r_1, r_2, \ldots r_m$.
- Мощности классов неизвестны, но обычно предполагается, что число элементов в классе превосходит длину выборки, взятой из этого класса.

Попытаемся найти прямым перебором такой набор номеров позиций $\alpha_1, \alpha_2, \ldots \alpha_k$, $k<n$ в словах, представляющих элементы множества $V = \{ v_1, v_2, \ldots v_N\}$, на котором система булевых функций $B_1, B_2, \ldots, B_m$, каждая связанная со своим классом, позволяла бы для слов, не принадлежащих выборкам, определять их принадлежность к кластерам элементов, близким к соответствующим выборкам по заданному критерию.

Конструируемая система булевых функций $B_1, B_2, \ldots, B_m$, которые называются элементарными классификаторами [7], должна обладать следующими свойствами: при подстановке любого слова $v=(x_1, x_2, \ldots, x_n) \subset V$ во все функции только одна из них должна принять значение единицы и ее номер будет считаться номером кластера, к которому близко проверяемое слово.

Переборный алгоритм состоит в отыскании такой последовательности позиций $\alpha_1, \alpha_2, \ldots \alpha_k$, $k<n$, которая удовлетворяла бы требованиям отсутствия пересечений кластеров логически близких элементами, принадлежащими разным выборкам.

Схема алгоритма построения кластеров элементов, близких к выборкам, состоит из выполнения следующих простых процедур: выбор, различных последовательностей номеров позиций и обработка каждой выбранной последовательности на предмет выяснения ее пригодности. Обработка каждой выбранной последовательности номеров позиций состоит в поочередном сравнении всех элементов выборки в означенных позициях на отсутствие совпадений с каждым элементом чужих выборок.

Заметим, что различных последовательностей номеров позиций, по которым будут строиться непересекающиеся кластеры, равно 2^n. Следовательно, число шагов перебора для построения одного кластера близких к выборке элементов ограничено данным числом. Шаг

перебора включает действия, которые необходимо выполнить на предмет определения пригодности выбранной позиции $\alpha_1, \alpha_2, \ldots \alpha_k$.

Отсюда следует, что полный перебор всевозможных последовательностей позиций и их обработка потребует не более $2^n \cdot R(r_1, r_2, \ldots r_m)$ машинных операций, где $R(r_1, r_2, \ldots r_m)$ - число операций, необходимых для анализа выбранных позиций .

Попробуем оценить значение $R(r_1, r_2, \ldots r_m)$, это позволит определить значение n – числа признаков в задаче кластеризации – с которыми справиться вычислительная система, заданной производительности.

Для простоты ограничимся рассмотрением случая двух выборок E_1, E_2, принадлежащие двум непересекающимся классам: K_1, K_2.

Анализ каждой выбранной позиции номеров $P=\{\alpha_1, \alpha_2, \ldots \alpha_k\}$, $k<n$ включает следующие действия:

Для каждой из рассмотренных выборок E_1, $E_2 \subset K_i$ последовательно перебираются её элементы (слова) и попарно сравниваются их значения в позициях $\alpha_1, \alpha_2, \ldots \alpha_k$ со значениями в этих позициях элементов выборок другого класса. Если не произошло совпадений значений в этих позициях ни с одним из элементов «чужой» выборки, то выбранный набор позиций обозначается как приемлемый для построения программы, конструирующей кластер элементов, близких к выборке $E_i \subset K_i$ по заданному критерию близости.

Если хотя бы одно парное сравнение дало совпадение одного из элементов выборки $E_i \subset K_i$ по набору $P=\{\alpha_1, \alpha_2, \ldots \alpha_k\}$, $k<n$ с каким-либо элементом из другой выборки по этому же набору, то такой набор признается непригодным (недопустимым), и производится переход к поиску другого набора позиций.

Если найден приемлемый набор $\alpha_1, \alpha_2, \ldots \alpha_k$, $k<n$, то он позволяет добавлять единицы в строки ТИ соответствующей булевой функции, отмечая тем самым элементы, логически близкие к исследуемой выборке. На основе полученной таким образом ТИ конструируется программа, способная определять, относится ли тестируемый элемент к кластеру элементов, близких к соответствующей выборке, расширяя её.

При заданном k кластеров, близких к данной выборке и не имеющих пересечений с чужими выборками, может быть несколько. Объединение таких кластеров дает единственный расширенный кластер элементов, близких по множеству наборов переменных длины k.

Программная реализация алгоритма кластеризации, очевидно, сводиться к выполнению сравнений значений в разрядах $\alpha_1, \alpha_2, \ldots \alpha_k$ найденного набора с элементами исходной выборки в этих разрядах.

Отметим, что число добавленных таким способом единиц в таблицу истинности булевой функции, связанной с рассматриваемой

выборкой, равно 2^{n-k}, в случае отсутствия других «хороших» наборов той же длины k. Следовательно, чем меньше k, тем больше по объему сконструированный кластер близких к выборке элементов.

Таким образом, для построения одного кластера близких к данной выборке E_i элементов по заданному критерию потребуется $r_1 \cdot r_2$ сравнений.

Поскольку максимальное число шагов перебора в поисках подходящего набора позиций равно 2^n, то для того, чтобы найти один требуемый набор позиций или убедиться в отсутствии искомой последовательности номеров позиций потребуется не более $2^n \cdot r_1 \cdot r_2$ сравнений.

Это крайне завышенная оценка трудоемкости перебора. Шаг перебора прекращается как только произошло совпадение значений по наборам переменных у двух разных выборок. Простой метод поиска на совпадение методом деления пополам может значительно сократить вычислительную сложность

Если производительность вычислительной системы оценивается 2^d операций сравнений в секунду, то оценка времени в секундах, которое потребуется для выполнения неоптимизированного переборного алгоритма выразится простой формулой

$$t \approx 2^{n-d} r_1 r_2$$

Если мы задались допустимым временем счета T_{max}, то должно быть выдержано соотношение: $2^{n-d} r_1 r_2 < T_{max}$, которое позволяет нам оценить n – число разрядов в двоичном слове, обозначающих признаки исследуемых объектов.

$$n \le \lg T_{\max} - \lg r_1 - \lg r_2 + d) \text{ **}$$

Если длины выборок r_1, r_2 равны 2^{10} ($\approx 10^3$), $T_{max}=2^{12}$ ($\approx$4000 сек.$\approx$1 час), d = 40 ($\approx 10^{15}$) (быстродействие машины в 1 терафлоп), то n должно быть меньше 32. Увеличение максимально допустимого времени расчетов на десятичный порядок прибавляет к допустимой размерности признаков тройку. Очевидно, что увеличение быстродействия вычислительной системы на тот же десятичный порядок также прибавляет к допустимой размерности признаков тройку

В практических приложениях размерность пространства признаков может достигать сотен двоичных разрядов. Формула ** позволяет оценить максимальное время , которое будет затрачено на кластеризацию в этом случае. Например, при n = 100 терафлопной машине для выполнения такого алгоритма конструирования одного кластера близких потребуется на обработку практически бесконечное число лет.

Отсюда понятна гонка за быстродействием, от терафлопных машин к петафлопным, затем к машинам с быстродействием 10^{18} и т. д.

Отсюда прямо следует, что для решения задач конструирования алгоритмов кластеризации данных с большой признаковой размерностью прямой перебор возможен только при анализе «малоразрядных» объектов и это оправдывает многочисленные исследования по сокращению признаковой размерности, применения идей обучения машин (ML) для отыскания приближенных решений подобного рода задач, в частности, методами генетического программирования.

После того как найдена допустимая последовательность $\alpha_1, \alpha_2, \ldots \alpha_k$, формула тестирующей булевой функции, которую следует реализовать машинной программой генетического программирования, определяющей близость элемента к выборке, строится как ДНФ по переменным $x_{\alpha_1}, x_{\alpha_2}, \ldots, x_{\alpha_k}$ в виде дизъюнкции конъюнкций:

$\sum_{j=1}^{r_i} \bigwedge_{i=1}^{k} x_{\alpha_i}^{\delta_{\alpha_i}^j}$, где j – номер элемента анализируемой выборки.

Программа, реализующая функцию, определяющую принадлежность любого элемента к кластеру близких к данной выборке, состоит в выяснении совпадает или нет тестируемый элемент с каким либо элементом чужой выборки в позициях $\alpha_1^i, \alpha_2^i, \ldots, \alpha_{k_i}^i$. Этот поиск на совпадение оптимизируется хорошо известными алгоритмами.

Проделав такого рода поиски подходящих наборов над выборками из классов, *и убедившись в том, что кластеры не пересекаются,* мы можем высказать гипотезу, что полученные кластеры входят в классы, которым принадлежат исходные выборки. Можно построить примеры, когда сконструированные таким образом кластеры действительно совпадают с классами, разделяющими исходное множество *V*.

О близости по структуре (пример).

Здесь мы рассмотрим сложность проверки критерия близости по максимуму совпадающих подслов, близкому к алгоритму парного сравнения двух последовательностей.

Шаблоном мы назовем двоичную последовательность длины *n*, где *n* число букв в элементах рассматриваемого множества *V*. Подсловом данного слова по шаблону мы назовем последовательность букв рассматриваемого слова, выбранных из позиций, соответствующих единицам шаблона. Длинной шаблона мы будем считать число единиц в нем. Соответственно, длиной подслова мы будем считать число выбранных по шаблону букв.

Пусть исходное слово s = (x_1, x_2, ... , x_n), пусть единицы шаблона расположены в разрядах с номерами $\alpha_1, \alpha_2, \ldots \alpha_k$, тогда

подслово, выбранное по данному шаблону будет представлять собой последовательность, букв длины k вида: $x_{\alpha_1} x_{\alpha_2} \ldots x_{\alpha_k}$. Это определение близости легко трансформируется на множества, состоящие из слов, построенных над любым конечным алфавитом.

Степень близости двух слов $s_1 = (x_1, x_2, \ldots, x_n)$ и $s_2 = (y_1, y_2, \ldots, y_n)$ по шаблонам одинаковой длины l определяется следующим образом. Выбирается шаблон для s_1 и другой шаблон для s_2, одинаковые по длине, так, чтобы подслова, выбранные по этим шаблонам, совпадали и были бы максимальными по длине. Такое определение близости эквивалентно тому, что нам следует найти минимальное число вычеркиваний букв из s_1 и s_2 так, чтобы полученные подслова были бы одинаковы.

При таком определении степени близости, её нахождение сводится к переборной задаче опробования всех возможных шаблонов как для слова s_1 так и для слова s_2. Порядок вычислительной сложности при этом оценивается величиной 2^{n-l} [5].

Во многих практических задачах классификации степень структурной близости регулируется условиями, налагаемыми на шаблоны. Если, например, потребовать одинаковости шаблонов для s_1 и s_2 [7], то степень близости будет величиной противоположной расстоянию по Хеменгу.

Если потребовать чтобы шаблоны для s_1 и s_2 отличались друг от друга только циклическим сдвигом, то переборная задача сведется к поиску максимальных по длине одинаковых подслов в s_1 и s_2. В задачах поиска совпадающих участков буквенных последовательностей, такое требование вполне оправдано.

Выбрав критерии определения структурной близости [6] можно на этом основании применить описанный выше алгоритм кластеризации («расширения выборок») близкими элементами для дополнения таблицы меток классов. По вычислительной сложности эти алгоритмы будут близки к оценкам, полученным при анализе кластеризации по логической близости.

Число необходимых сравнений известными приемами оптимизации может быть значительно уменьшено, правда, за счет накладных расходов по предварительной сортировки выборок [11].

Замечание.

Современные вычислительные системы представляют собой мультипроцессоры с массовым параллелизмом, алгоритмы, описанные выше, сравнительно легко распараллеливаются [13,15]. При более тщательном анализе возможности решения задач построения классификаторов с использованием переборных алгоритмов следует

учитывать то, что от пиковой производительности этих машин в среднем используется обычно не более 20%.

Изложенные выше алгоритмы позволяют строить расширенные таблицы меток классов, которую можно рассматривать как табличное задание некоторой целочисленной функции, заданной в некоторых точках её определения.

Поскольку нейронные сети являются универсальным инструментом аппроксимации функций [12], можно надеется, что их применение позволит построить функции-классификаторы, основанные на введенных критериях близости, как средство предварительной обработки выборок. Учитывая вышесказанное, можно по методике, приведенной выше, оценить приемлемые значения длины вектора признаков, которые возможно обработать на машинах заданной производительности за заданное время.

Литература

1. Srinivas M, Patnaik L. Genetic Algorithms, a survey. //J. Computer, v. 27, № 6, 28-43, IEEE press. June 1994.
2. Forrest S. Genetic Algorithm: Principles of Natural Selection Applied to Computation.// J. Sciense. V. 261, 872-878, August 1993.
3. Ю. И. Журавлев. Непараметрические задачи распознавания образов. // Ж. Кибернетика, N6, Москва 1976.
4. Иванов С.А. Методы поиска закономерностей в символьных последовательностях // Программные системы и инструменты. Тематический сборник № 8, М.: Изд-во факультета ВМиК МГУ, 2007.
5. Magnusson M.S. Discovering Hidden Time Patterns in Behavior: T-Patterns and their Detection // Behavior Research Methods, Instruments and Computers. 2002. 32. N 1. P. 93-100.
6. Яблонский С. В. Введение в дискретную математику.// М.,Высшая школа, 2003 г.
7. Дюкова Е. В. Дискретные (логические) процедуры распознавания: принципы конструирования, сложность реализации и основные модели. Типография МПГУ, М., 2003 г.
8. Сачков В. Н., Комбинаторные методы дискретной математики. // М., 1977.
9. Ту, Дж., Гонсалес, Р. – Принципы распознавания образов: Пер. с англ. // М.: Мир, 1978.
10. Holland J. Adaptation in natural and artificial systems. // MIT press, Cambridge MA, 1992
11. Forrest S. Genetic Algorithm: Principles of Natural Selection Applied to Computation.// J. Sciense. V. 261, 872-878, August 1993.
12. С. Осовский. Нейронные сети для обработки информации. // М.: «Финансы и статистика», 2002.

13. R. Miller, L. Boxer. Algorithms Sequential & Parallel, a unified approach.// PrenticeHall, 2000.
14. С. Осовский. Нейронные сети для обработки информации. // М., «Финансы и статистика», 2002.
15. Quinn M.J. Designing Efficient Algorthms for Parallel Computers. //McCraw-Hill. 1987.
16. Рудаков К.В., Чехович Ю.В. О проблеме синтеза обучающих алгоритмов выделения трендов (алгебраический подход) // Прикладная математика и информатика №8, Из. ВМК МГУ, 2001 г., стр.97-114.
17. Коваленко Д.С., Костенко В.А., Васин Е. А. Исследование применения алгебраического подхода к анализу временных рядов. // Методы и средства обработки информации. Из. Факультета ВМК МГУ, 2005г., стр. 553-559.
18. Математическая энциклопедия. // Изд.«Советская энциклопедия», том 3 М., 1982 г.
19. Srinivas M, Patnaik L. Genetic Algorithms, a survey. //J. Computer, v. 27, № 6, 28-43, IEEE press. June 1994.
20. Ribeiro J, and al. Genetic Algorithms. Programming Environments. //Computer, v. 27, № 6, 28-43, 17-26, IEEE press, June, 1994.
21. Forrest S. Genetic Algorithm: Principles of Natural Selection Applied to Computation.// J. Sciense. V. 261, 872-878, August 1993.
22. John Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection //MIT Press 1992.
23. John Koza, Genetic Programming II: Automatic Discovery of Reusable Programs //MIT Press,1994.
24. Ю.И. Журавлев. Об алгоритмах распознавания с представительными наборами (о логических алгоритмах) // Журнал вычислительной математики и математической физики, 2002, Т. 42, 9, С. 1425-1435 .
25. Чехович Ю.В. Применение алгебраического подхода к задачам выделения трендов. // Матем. методы распознавания образов (ММРО-10). Докл. 10-й Всероссийск. конф.ВЦ РАН. М.: АЛЕВ-В, 2001. С.315-316.

Раздел II
Параллельное программирование

Джосан О.В., Попова Н.Н.

Метод сжатия изображений для визуализации на массивно-параллельных вычислительных системах

Введение

В работе рассмотрен метод параллельного сжатия изображений и видео для системы визуализации результатов научных вычислений, проводимых на терафлопсных и петафлопсных вычислительных системах. .

Визуализация результатов научных вычислений, проводимых на массивно-параллельных вычислительных системах(МПВС), обладает рядом особенностей. Получаемый в вычислительных экспериментах объем данных может быть очень большим. Например, при моделировании турбулентного горения на вычислительной системе Blue Gene /P для проекта FLASH с использованием 8000 четырехядерных вычислительных узлов генерируется 16Gb данных каждые 10-15 минут, общий объем данных эксперимента составляет 300Tb[1]. Объем данных, получаемых в задаче климатического моделирования, составляет 345Tb[2]. Объем данных, получаемых в результате проведения экспериментов на системе Blue Gene /P, установленной в МГУ, измеряется сотнями гигабайт. Система визуализации результатов научных вычислений на МПВС должна поддерживать возможность работы с данными большого объема.

Сжатие получившихся при визуализации данных является важным методом в системе визуализации, т.к. позволяет существенно сократить объем передаваемых данных. Возможно использование нескольких процессоров для сжатия, что позволяет осуществлять работу по сжатию видеопоследовательности в параллельном режиме.

В данной работе для сжатия изображений и видеопоследовательностей предложен новый метод кодирования, основанный на блочном кодировании изображения. При этом используется быстрый способ оценки сложности блоков изображения, что позволяет оценить временные затраты на кодирование и осуществить разбиение блоков по процессорам в зависимости от этого времени, чтобы минимизировать общее время кодирования кадра. Каждое изображение или кадр видеопоследовательности кодируется как последовательность блоков, при этом для каждого блока подбирается

оптимальный способ его кодирования в зависимости от посчитанной визуальной сложности.

Применение блочного кодирования для метода параллельного сжатия изображений и видеопоследовательностей продиктовано необходимостью разделения по данным: таким образом, чтобы блоки изображения могли бы обрабатывается независимо на разных процессорах. Однако использование блочного кодирования чревато появление блочных артефактов. Поэтому используемый размер блока должен быть небольшим.

Блок-схема метода сжатия

Блок-схема предложенного алгоритма кодирования показана на Рис.1. Размер используемого блока – 4x4 пикселя.

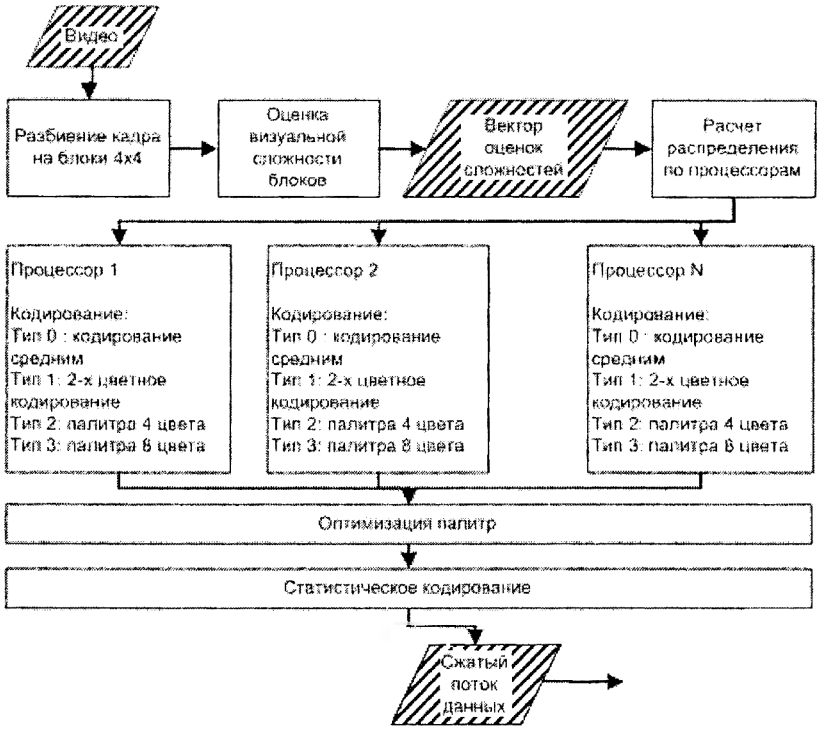


Рис.1 Блок-схема параллельного метода сжатия изображений для системы визуализации

Для оценки визуальной сложности блока строится исходя из частотных характеристик блока. Оцениваются различные

характеристики блока, которые показывают насколько близкие значения у соседних пикселей в блоке. Пиксели в блоке обрабатываются в формате YCbCr. Преобразование из RGB формата осуществляется по стандартным формулам[3]:

$$Y = 0.257R + 0.504G + 0.098B + 16$$
$$Cb = -0.148R - 0.291G + 0.439B + 128$$
$$Cr = 0.439R - 0.368G - 0.071B + 128$$

Обратное преобразование при декодировании осуществляется по формулам:

$$R = 1.164(Y - 16) + 1.596(Cr - 128)$$
$$G = 1.164(Y - 16) - 0.813(Cr - 128) - 0.392(Cb - 128)$$
$$B = 1.164(Y - 16) + 2.017(Cb - 128)$$

Оценка визуальной сложности блока

Предложено оценивать визуальную сложность блоков с помощью следующего интегрального соотношения:

$$F = \frac{\sigma}{12}\left(\frac{D_x}{4} + \frac{D_Y}{4} + \frac{D_{d1}}{3} + \frac{D_{d2}}{3}\right) + \frac{\sigma}{2}\sum_{T\in\{Y,Cb,Cr\}}\left(A_T + B_T\right)$$

где $\sigma, D_X, D_Y, D_{d1}, D_{d2}, A_T, B_T$ - соотношения пикселей в блоке, рассчитанные с помощью соотношений, приведенных далее.

Рассчитывается дисперсия в блоке и оптимальные цвета для двухцветного кодирования:

$$\bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i, \quad \sigma = \frac{1}{k}\sum_{i=1}^{k}(x - x_i)^2, \quad a = \bar{x} - \sigma \times \sqrt{\frac{q}{k-q}}, \quad b = \bar{x} + \sigma \times \sqrt{\frac{k-q}{q}}$$

где k – это количество пикселей в блоке, q – это количество пикселей в блоке, которые больше среднего значения в этом блоке. Проводится оценка количества и резкости границ в блоке с помощью следующих формул:

$$D_X = \sum_{\substack{k=-1..1\\ l=-1..2}} \left| I_Y(i+k+1, j+l) - I_Y(i+k, j+l) \right|$$

$$D_Y = \sum_{\substack{k=-1..2\\ l=-1..1}} \left| I_Y(i+k, j+l+1) - I_Y(i+k, j+l) \right|$$

$$D_{d1} = \sum_{\substack{k=-1..1\\ l=-1..1}} \left| I_Y(i+k+1, j+l+1) - I_Y(i+k, j+l) \right|$$

$$D_{d2} = \sum_{\substack{k=-1..1\\ l=-1..1}} \left| I_Y(i+k+1, j+l) - I_Y(i+k, j+l+1) \right|$$

Дополнительно для оценки сложности проводится расчет параметров частотности области по формулам:

$$A_T = \frac{1}{4} \sum_{\substack{h \in \{-1,1\} \\ g \in \{-1,1\}}} \left| I_T(i, j) - I_T(i + h, j + g) \right|$$

$$B_T = \frac{1}{4} \left(\left| I_T(i, j-1) - I_T(i-1, j) \right| + \left| I_T(i-1, j) - I_T(i, j+1) \right| + \right.$$

$$\left. \left| I_T(i, j+1) - I_T(i+1, j) \right| + \left| I_T(i+1, j) - I_T(i, j-1) \right| \right)$$

где T – это цветовой канал, I(i,j) – центральный пиксель в блоке.

Далее определяется, какому из заданных диапазонов значений принадлежит получившееся значение F. Номер диапазона соответствует номеру метода кодирования, который оптимален для блока. В текущей реализации количество возможных методов кодирования M=4. Однако это значение может быть расширено: возможно добавление и замена методов кодирования.

Для оптимизации времени вычислений предложен метод оптимального распределения блоков по вычислительным процессорам, основанный на оценке визуальной сложности блока. На каждом процессоре соответствующие блоки кодируются с помощью одного из 4-х методов: кодирование средним, двухцветное кодирование, кодирование с помощью 4-х цветной палитры, кодирование с помощью 8-х цветной палитры. Предложенный метод кодирования описан в работах [4,5]. Для оптимизации распределения блоков по вычислительным узлам используется равномерное распределение блоков в зависимости от их сложности. Для этого блоки сортируются в порядке значений из визуальных сложностей, затем последовательно распределяются по процессорам.

Методы кодирования блоков

Рассмотрим более подробно предложенные методы кодирования блоков. Первый метод кодирования требует наименьшее количество бит для кодирования блока и имеет наименьшую вычислительную сложность, но при этом вносится много визуальных искажений. При использовании этого метода пиксели блока кодируются средним значение для пикселей этого блока. Это значение подсчитано на этапе оценки сложности блока. Таким образом, количество бит, требуемое для кодирования в данном случае, равно количеству бит, которым передается значение одного цвета. При этом самый младший бит не передается, а при декодировании восстанавливается нулем, поэтому в случае 256 цветов, требуется семь бит.

Следующий метод кодирования – это стандартный метод BTC кодирования, описанный в [6,7]. Основная идея этого метода состоит в том, что для кодирования блока используются два цвета. Эти два цвета определяются по формулам:

$$a = \bar{x} - \sigma \times \sqrt{\frac{q}{k-q}}, \quad b = \bar{x} + \sigma \times \sqrt{\frac{k-q}{q}}$$

где q – это количество пикселей в блоке, значение которых больше, чем среднее значение $\bar{x}$, σ – значение дисперсии.

Рис.2 Методы определения оптимальной палитры для блока

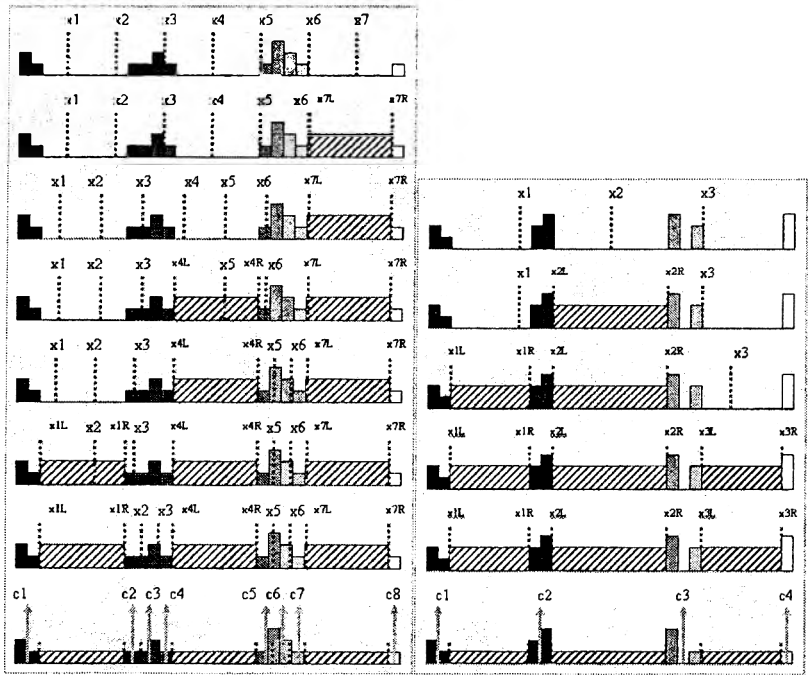
При этом в качестве закодированной последовательности передается среднее значение $\bar{x}$, дисперсия σ и битовая карта для пикселей, где бит, соответствующий пикселю, равен единице, если значение пикселя больше, чем среднее значение для блока, и равно нулю иначе. При этом $\bar{x}$, σ уже посчитаны на шаге оценки сложности блока. Этот метод требует 28 бит для передачи блока: 16 бит на

битовую карту, 8 бит для передачи среднего значения и 4 бита для передачи дисперсии. Этот метод используется для кодирования блоков с большей визуальной сложностью, чем у блоков, кодируемых первым методом.

Еще один метод, который используется для кодирования, - это определение оптимальной палитры. Для блока определяется оптимальная палитра и карта, в которой указаны номера цветов палитры, которые наиболее близки к цветам пикселей блока. Предложен метод определения оптимальной восьмицветной и четырехцветной палитры. Критерий оптимальности выбран следующий: минимизировать максимальную разность между цветами закодированного блока и реальными цветами блока. Для достижения этого критерия предложено минимизировать количество нулевых интервалов в гистограмме и цвета палитры распределять между оставшимися частями гистограммы.

Схема этого метода кодирования показана на Рис.2 и процесс определения оптимальной палитры проиллюстрирован на Рис.3а для восьмицветной палитры, на Рис.3б для четырехцветной палитры. Палитра определяется по распределению цветов в исходной гистограмме.

Рассмотрим применение алгоритма построения оптимальной палитры из 8 цветов на примере. На первом шаге определяют гистограммы блока. Далее выбирают рабочий интервала [x0, x8], где x0 является минимальным ненулевым значением гистограммы, x8 является максимальным ненулевым значением гистограммы. На следующем шаге рабочий интервал разбивают на восемь интервалов одинаковой длины точками множества A = {x1, x2, x3, x4, x5, x6, x7}. Затем проверяют, принадлежат какие-либо точки xi к нулевому интервалу и, если такие точки имеются, выбирают точку множества A с наименьшим индексом, принадлежащей самому длинному нулевому интервалу, с последующим исключением xi из множества A. Далее точку xi преобразуют в две точки – xiL и xiR – наиболее близкие к xi слева и справа ненулевые значения в гистограмме. Слева и справа от этих двух новых точек соответственно получаются два новых рабочих интервала гистограммы. Оставшиеся точки множества A распределяют в этих двух интервалах, проводя равномерное разбиение интервалов этими точками: точки с номерами меньше i распределяют в левом интервале, а точки с большими чем i номерами индексов – в правом интервале. Далее снова выбирают и преобразовывают в интервал точку из множества A, которая попала в наибольший нулевой интервал, если такая точка может быть выбрана. Этот процесс повторяется до исчерпания точек во множестве A, попадающих в нулевые интервалы на гистограмме. Далее определяют оптимальную палитру как середины восьми получившихся интервалов.



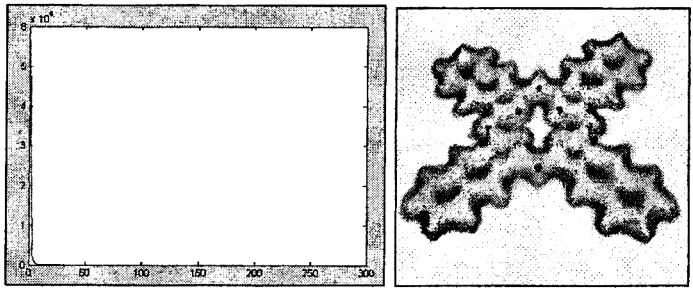
а) б)

Рис.3 Пример расчета оптимальной палитры для блока: а) 8 цветов; б) 4 цвета

Получившуюся цветовую палитру преобразуют во множество разностей между соседними цветами, при этом палитру изменяют так, чтобы все разности стали меньше 32. Т.е. в худшем случае получается равномерное распределение цветов палитры по гистограмме. Однако для большинства изображений и видеопоследовательностей значение получившейся статистики показывают, что соседние цвета находятся недалеко друг от друга. На Рис.4а показана статистика разностей цветов в палитрах блоков изображения, показанного на Рис.4б. Как видно из графика большинство значений разностей изначально не превышают значения в 32 градации, поэтому преобразование палитр затронет только незначительное количество блоков, качество которых не повлияет на общее визуальное качество картинки.

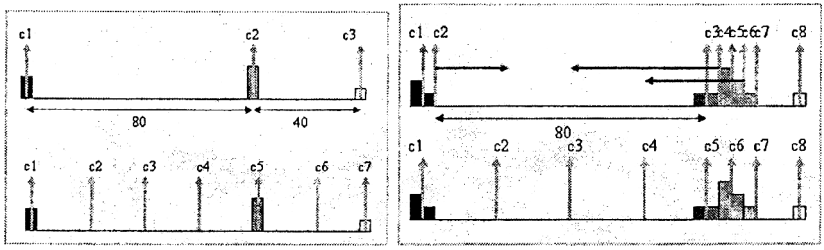
На первом шаге для изменения палитры добавляют фиктивные цвета, если это можно сделать. В отношении фиктивного цвета сначала вычисляют, является ли число требующихся фиктивных цветов меньше или равно числу имеющихся фиктивных цветов (если количество использованных цветов в палитре меньше восьми), если число фиктивных цветов достаточно, то их равномерно распределяют по длинным интервалам, если это не достаточно, то разбивают рабочий

интервал гистограммы на восемь одинаковых интервалов и рассматривают середины этих интервалов в качестве палитры. Рис.5а иллюстрирует предложенную идею добавления фиктивных цвстов. После добавления фиктивных цветов, в случае если остались разности больше 32 градаций, проводится смещение цвстов. Перестановка цветов проиллюстрирована на Рис.5б.



а) б)

Рис.4 статистика разностей в цветах палитр



а) б)

Рис.5 а) Добавление фиктивных цветов в палитру; б) Перестановка цветов в палитре

После определения цветовой палитры вычисляют цветовую карту, в которой номер цвета на палитре, ближайшей к реальному значению пикселя, является значением пикселя в цветовой карте. Аналогичный алгоритм применяется для определения четырехцветной палитры. Кодирование восьмью цветами требует 91 бит, кодирование четырьмя цветами – 64 бита. Эта длина закодированного пакета включает в себя два бита, которые определяют тип кодирования для блока, по которому на стороне декодера будет осуществлено восстановление изображения.

Статистическое кодирование подразумевает сжатие без потерь потока информации. Для этого применяется сворачивание цепочек нулей RLE [8].

Анализ эффективности предложенного метода сжатия

Для измерения количественных характеристик эффективности предложенного метода сжатия рассмотрим следующий модельный эксперимент по моделированию молекулярных переключателей с помощью кода CPMD. В рассматриваемой модели использовалось 82 атома, 129 орбиталей. Объем данных эксперимента составил около 200Gb. Для счета использовалось 512 процессоров, при этом время расчета составляет около суток. Для эксперимента использовались первые 1024 файла исходных данных, представленных в наборе файлов формата .cube, каждый из которых описывает положение молекулы в момент времени.

На тестовом примере проанализирована эффективность алгоритма сжатия. Размер получаемого изображения по одному файлу до применения алгоритма сжатия составлял от 120K для изображения 200x200 до порядка 2M для изображения 800x800. На Рис.6 представлен график степени сжатия для нескольких кадров рассматриваемой видеопоследовательности. График показывает степень сжатия для изображения размером 200x200 для качества без визуальных потерь с использованием предложенного многопоточного метода сжатия. Из графика видно, что степень сжатия меняться. В среднем степень сжатия равна 10. Таким образом, исходный размер данных вычислительного эксперимента, используемых при данном модельном эксперименте, составлял порядка 31 гигабайта. Размер полученной в этом эксперименте видеопоследовательности до применения сжатия составил 123M, размер полученной видео последовательности после применения алгоритма сжатия составил около 12M.

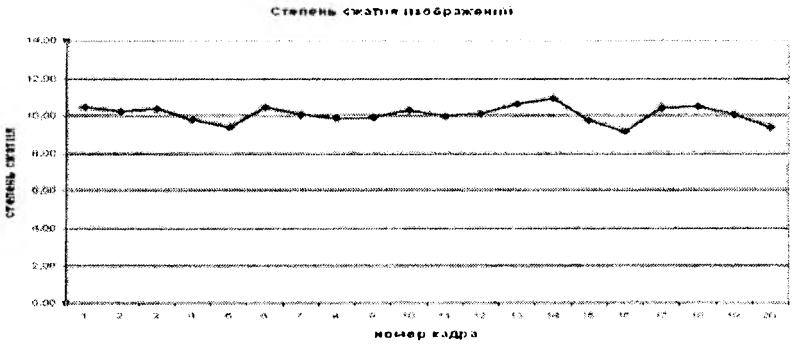


Рис. 5 Степень сжатия модельного эксперимента

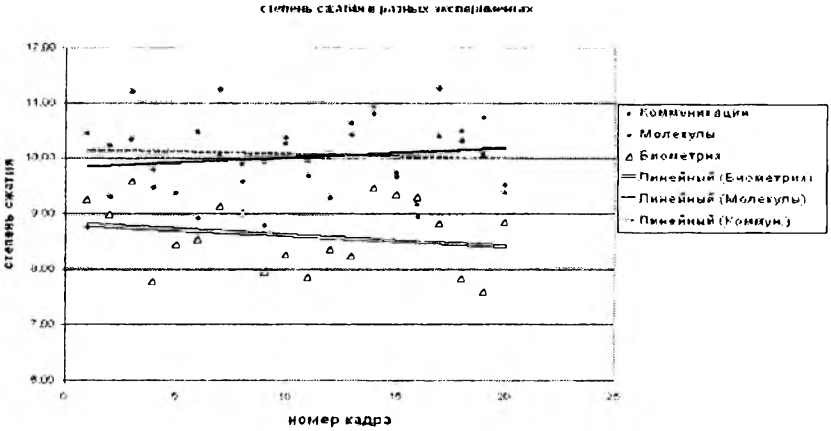


Рис. 6 Сравнение полученных степеней сжатия в различных экспериментах

На Рис.6 показано сравнение степеней сжатия для применения визуализации в реальных данных прикладных экспериментов.

Заключение

Разработан специализированный метод многопоточного сжатия изображений, который основан на блочном кодировании кадра и использует специализированную функцию оценки визуальной сложности блока перед распределением блоков для кодирования на нескольких процессорах. Предложена соответствующая функция оценки визуальной сложности блоков, основанная на частотных характеристиках. Предложенный метод сжатия изображений позволяет получить существенное улучшение качества сжатого изображения по сравнению со стандартными методами сжатия. Метод учитывает особенности входного изображения, а именно, тот факт, что сжатию подвергаются синтезированные изображения, которые включают в себя большие однородные области фона, объекты с ярко выраженными границами и высокочастотные области с большим количеством деталей. Эксперименты показали, что предложенный метод позволяет сжимать изображения с точностью максимум до 4 градаций по сравнению с 20 градациями, которые дает стандартный mpeg2 кодек при той же степени сжатия. Получено в среднем десятикратная степень сжатия при качестве без визуальных потерь.

Работа выполнена при поддержке грантов РФФИ 08-07-12081 и 08-07-00445-а.

Литература

1. T. Peterka, R. B. Ross, H.-W. Shen, K.-L. Ma, W. Kendall, and H. Yu, "Parallel Visualization on Leadership Computing Resources," Preprint ANL/MCS-P1656-0709, July 2009.
2. T. Peterka, R. Ross, H. Yu, K.-L. Ma, W. Kendall, and J. Huang, "Assessing and Improving Large Scale Parallel Volume Rendering on the IBM Blue Gene/P," Preprint ANL/MCS-P1554-1008, October 2008
3. Keith Jack , YCbCr to RGB Considerations , http://www.intersil.com/data/an/AN9717.pdf
4. Мишуровский М.Н. Джосан О.В. Рычагов М.Н. Способ компактного представления цветных цифровых изображений, предназначенный для применения в системах цифровой обработки видеосигналов в реальном времени // тезисы конференции "Телевидение: передача и обработка изображений", Россия, Санкт-Петербург, 2009
5. Джосан О.В., Мишуровский М.Н. Анализ методов сжатия цифровых цветных изображений без визуальных потерь // тезисы конференции "Телевидение: передача и обработка изображений", Россия, Санкт-Петербург, 2008
6. Zeng, B. Wang, Q. Neuvo, Y. , BTC image coding using two-dimensional median filter roots // Circuits and Systems, , IEEE International Sympoisum, vol.1, pp. 400-403, 1991
7. Zeng, B.; Neuvo, Y., Interpolative BTC image coding with vector quantization // Communications, IEEE Transactions on Volume 41, Issue 10, Oct. 1993, pp.1436 – 1438
8. Arturo San Emeterio Campos, Run Length Encoding, http://www.arturocampos.com/ac_rle.html

Глазкова Е.А., Попова Н.Н.

Исследование влияния виртуальных топологий MPI и способа назначения процессов на процессоры на эффективность выполнения гибридной программы на примере системы Blue Gene/P

Введение

В настоящее время существует большой разрыв между пиковой производительностью суперкомпьютеров и той реальной производительностью, которую они показывают на тестовых пакетах и пользовательских задачах. На тестовом пакете Linpack суперкомпьютеры обычно достигают примерно 60-80% от пиковой производительности [1]. На пользовательских задачах производительность обычно составляет всего 15-20% от пиковой. Кроме того, суперкомпьютеры обычно разрабатываются для решения определенного класса задач, потому при решении других типов задач их ресурсы могут использоваться гораздо менее эффективно.

Существуют различные подходы к созданию параллельных программ. Одними из наиболее распространенных подходов являются использование библиотеки MPI для систем с распределенной памятью и интерфейса OpenMP для систем с разделяемой памятью.

Современные компьютеры для высокопроизводительных вычислений, как правило, многоядерные: каждый узел в них представляет собой систему с разделяемой памятью. Скорее всего, именно многоядерность останется основной тенденцией в развитии процессоров в ближайшее время. Использование многоядерных архитектур и SMP-кластеров в высокопроизводительных вычислениях требует разработки для них эффективных методов программирования [2]. Одним из подходов к созданию параллельных программ для таких архитектур является гибридный (MPI + OpenMP) подход.

Гибридная модель программирования MPI + OpenMP потенциально может увеличить производительность многопроцессорных систем, построенных на основе многоядерных процессоров, по сравнению с традиционно используемыми MPI реализациями, так как она содержит несколько уровней параллелизма и учитывает иерархическую структуру памяти.

Как отмечалось в работах [2], [3], [4], эта модель может частично устранить некоторые недостатки модели MPI. К числу таких недостатков относится слабая масштабируемость приложения с возрастанием числа MPI-процессов, которую вызывают большие накладные расходы на передачу сообщений и нарастающая с

увеличением числа MPI-процессов динамическая несбалансированность их нагрузки. Этот недостаток частично компенсируется укрупнением MPI-процессов и уменьшением их числа за счет использования OpenMP-нитей с динамической балансировкой нагрузки. Также гибридная модель позволяет частично избежать дублирования данных на одном узле, когда несколько OpenMP-нитей разделяют один большой массив.

Но, несмотря на свои потенциальные достоинства, гибридная модель имеет целый ряд трудностей. Достаточно часто использование гибридной модели программирования приводит, наоборот, к уменьшению эффективности.

Выбор между чистым MPI и гибридным MPI + OpenMP подходом при решении конкретной задачи является очень нетривиальным. Следует учитывать влияние многих параметров, определяемых архитектурой машины, таких как стоимость коммуникаций, способ обращения к памяти и баланс производительности между процессором, памятью и коммуникационной средой [5], [6], [7], [8].

Для того чтобы гибридные программы выполнялись эффективно, необходимо понимать, какие параметры влияют на время выполнения и возможность масштабируемости. В системе Blue Gene/P имеется ряд параметров настройки, которые могут влиять на производительность системы и значениями которых пользователь может управлять. Такими параметрами являются режим выполнения, определяющий соотношение числа MPI-процессов и OpenMP-нитей на вычислительном узле (Рис. 1), и mapping – способ назначения MPI-процессов на процессоры системы (Рис. 2). Настраиваемые параметры присутствуют и в самой тестовой задаче (способ распределения данных и виртуальная топология).

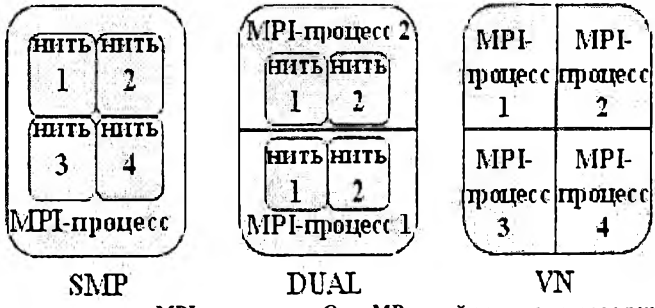


Рис.1 Соотношение числа MPI-процессов и OpenMP-нитей для различных режимов выполнения программы в системе Blue Gene/P.

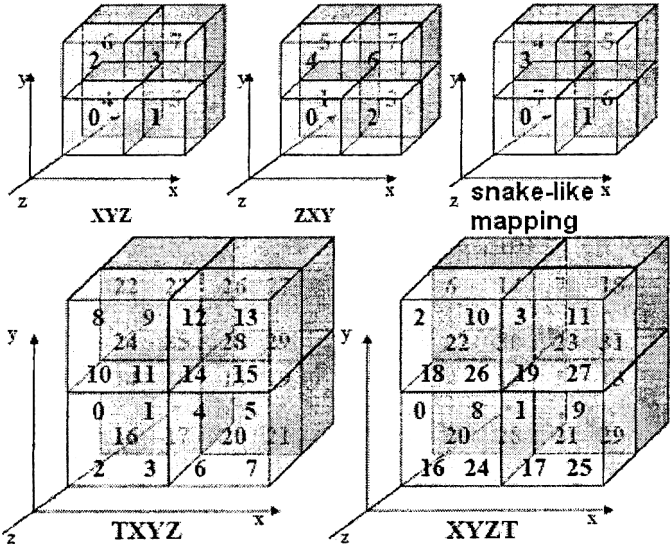


Рис. 2. Способы назначения MPI-процессов на процессоры.
Вверху – для режима SMP (координата T, соответствующая номеру ядра в вычислительном узле, равна 0 для всех MPI- процессов).
Внизу – для режимов VN и DUAL.

Постановка задачи

Целью работы является разработка методов настройки эффективности гибридной параллельной программы на примере модельной задачи (решения разностной задачи Дирихле методом Якоби в трехмерной области (параллелепипед)). Предполагается, что исследование даст возможность определить, насколько сильно и каким образом виртуальная топология MPI и способ назначения процессов на процессоры (mapping) влияют на производительность параллельной программы и возможность ее масштабируемости, и выработать рекомендации для пользователей по выбору значений этих параметров.

Метод Якоби решения разностной задачи для уравнения Лапласа – это типичная тестовая задача в параллельном программировании, так как в параллельной реализации метод Якоби включает в себя общие шаблоны для большинства параллельных задач: вычисления, взаимодействия между соседями и глобальную редукцию. Исходя из особенностей межпроцессорных взаимодействий в реализации метода, он должен хорошо ложиться на архитектуру трехмерной решетки.

Метод Якоби построен на шаблонных вычислениях: каждая точка трехмерной сетки обновляется с использованием взвешенной

суммы некоторого подмножества соседних с ней точек. Этот вид вычислений характерен для многих научных приложений, в которых требуется решать уравнения в частных производных. Шаблонные вычисления применяются как в простых методах, таких как метод Якоби, так и в сложных современных адаптивных методах. К сожалению, производительность для такого вида вычислений обычно далека от пиковой из-за дисбаланса в скорости работы процессора и оперативной памяти, который проявляется из-за использования в вычислениях больших структур данных, размеры которых обычно превосходят размер кэша.

Понимание влияния параметров системы и задачи на время выполнения программы для метода Якоби могло бы помочь выработать рекомендации для настройки этих параметров для программных реализаций других численных методов, основанных на шаблонных вычислениях на сетке.

Краткое описание гибридной реализации алгоритма

В алгоритме используются два буфера с данными. Один - для значений, полученных на предыдущей итерации, другой - для значений вычисляемых на новой итерации, используя результаты предыдущей.

Перед выполнением алгоритма нужно записать в один из буферов начальное приближение и установить порядок буферов. На каждой итерации алгоритма вычисляется погрешность Δ. Алгоритм останавливается, когда Δ становится меньше некоторого числа или после выполнения заданного числа итераций.

Важный момент при организации вычислений состоит в том, что на процессор, выполняющий обработку какого-либо блока данных, должны быть продублированы боковые грани соседних с ним блоков.

```
// действия, выполняемые на каждом процессоре
do {
 // обмен «теневыми» гранями с соседями
 // обработка блока
 // вычисление общей погрешности вычислений
}while (Δ < eps );
```

Рис. 3. Основной цикл программы.

Из-за того, что данные хранятся в памяти не непрерывно, для выполнения межпроцессорных обменов предварительно формируются передаваемые сообщения. Потом при помощи функций MPI_Cart_shift и MPI_Sendrecv происходит определение ранга процесса, которому надо послать данные, и процесса, от которого данные необходимо получить, и обмен данными с соседними процессами. Для вычисления Δ

производится глобальная редукционная операция при помощи функции MPI_Allreduce.

На рисунке 3 представлена схема используемого алгоритма.

Целевая архитектура

В качестве целевой архитектуры, на которой проводится исследование, используется система Blue Gene/P. Базовыми элементами системы Blue Gene/P являются вычислительные узлы, состоящие из четырех ядер PowerPC 450, работающих над общей оперативной памятью размером 2Гб. Ядро PowerPC 450 – это 32 битный процессор с частотой 850 МГц и пиковой производительностью 3.4 GLOPS. Таким образом, производительность одного вычислительного узла, состоящего из четырех ядер, 13.6 GFLOPS. Вычислительные узлы соединены между собой несколькими сетями, в том числе трехмерным тором (сетью для взаимодействий точка-точка), сетью для коллективных операций и сетью для барьерной синхронизации [9].

Система Blue Gene/P имеет иерархическую структуру, для которой гибридное программирование потенциально может дать выигрыш в производительности.

Во многих параллельных приложениях линейная нумерация процессов не соответствует логической структуре задачи. Часто в задачах процессы упорядочиваются в логические структуры двух или трехмерной решетки. Виртуальные топологии MPI предоставляют удобный механизм именования процессов в группе внутри коммуникатора.

Необходимо четко разграничивать виртуальную топологию MPI и физическую топологию системы. Виртуальная топология MPI является машинно-независимой характеристикой приложения. Mapping – отображение виртуальной топологии на физическую топологию системы является машинно-зависимым процессом и выполняется пользователем с помощью изменения настроек на целевой системе. Отображение виртуальной топологии на физическую топологию системы находится вне компетенции MPI [10]. Удачный выбор виртуальной топологии для заданной физической топологии системы может существенно улучшить выполнение коммуникаций на целевой системе.

В тестовой задаче виртуальная топология MPI тесно связана со способом распределения данных. Виртуальная топология определяет число разбиений вдоль каждого измерения задачи и, значит, размер коммуникаций.

Модельная задача имеет виртуальную MPI-топологию решетки, которая должна хорошо соответствовать физической топологии процессоров системы Blue Gene/P.

По умолчанию назначение MPI-процессов на процессоры в системе Blue Gene/P происходит в порядке XYZT, где (X, Y, Z) - координаты вычислительного узла в трехмерном торе, T – номер ядра внутри вычислительного узла. Назначением MPI-процессов на процессоры можно управлять двумя способами: задавать одну из 12 стандартных перестановок X, Y, Z, T или указывать собственный файл с отображением.

В ходе работы был проведен ряд экспериментов на системе Blue Gene/P, чтобы исследовать время работы тестовой программы при различных значениях перечисленных параметров.

Методика проведения тестирования

Первая серия экспериментов проводилась на минимальном доступном в системе размере партиции - 128 вычислительных узлов (512 ядер), и при размере задачи 2304*1280*2560 элементов типа double. Такой размер задачи был выбран, как компромисс между необходимостью максимально использовать память вычислительного узла и желанием иметь в качестве размера задачи числа, делящиеся нацело на степени двойки, что позволило бы разделить данные между узлами блоками одинакового размера. Первая серия экспериментов включала в себя сначала исследование и анализ времени выполнения программы при распределении данных полосами, а затем аналогичный эксперимент для распределения данных блоками. Для параметров системы и задачи, соответствующих наиболее характерным результатам экспериментов, был проведен более глубокий анализ с использованием библиотеки для профилирования MPI-приложений mpiP [11].

Вторая серия экспериментов состояла в исследовании влияния некоторых избранных параметров на масштабируемость приложения. Под масштабируемостью понималась способность приложения при увеличении размеров задачи, пропорциональном увеличению числа процессоров, сохранять прежнее время выполнения.

Результаты тестирования

Результаты первой серии экспериментов показали, что исслсдуемые параметры (способ распределения процессов по процессорам, виртуальная топология MPI и режим выполнения программы в системе Blue Gene/P) достаточно сильно влияют на время выполнения программы. Разница между худшим временем (67.6854 сек), полученном при режиме выполнения VN, случайном назначении процессов на процессоры и виртуальной топологии 1*1*512, и лучшим временем выполнения программы (34.4916 сек), полученном при VN режиме, порядке TYXZ назначения процессов на процессоры и виртуальной топологии 8*4*16, составляет примерно 1.96 раза. На

рисунках 4 и 5 показано влияние виртуальной топологии MPI на время выполнения программы.

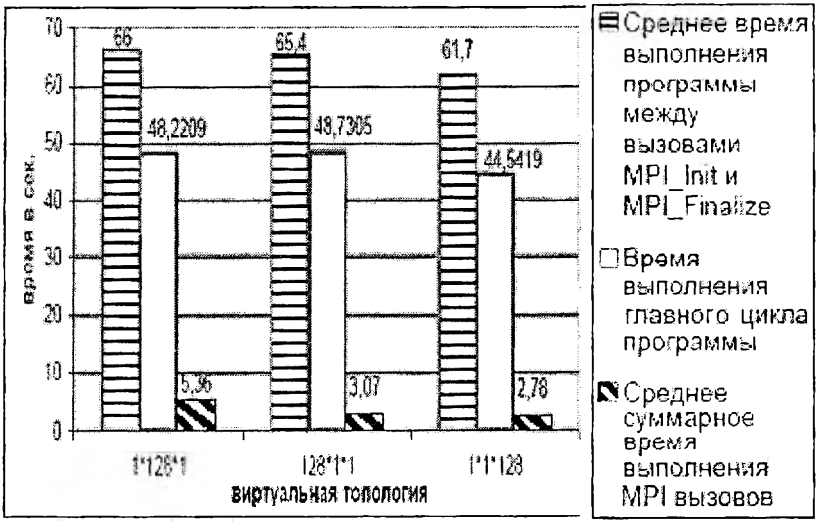


Рис. 4. Среднее время выполнения для различных виртуальных топологий в режиме SMP на 128 вычислительных узлах при распределении данных полосами.

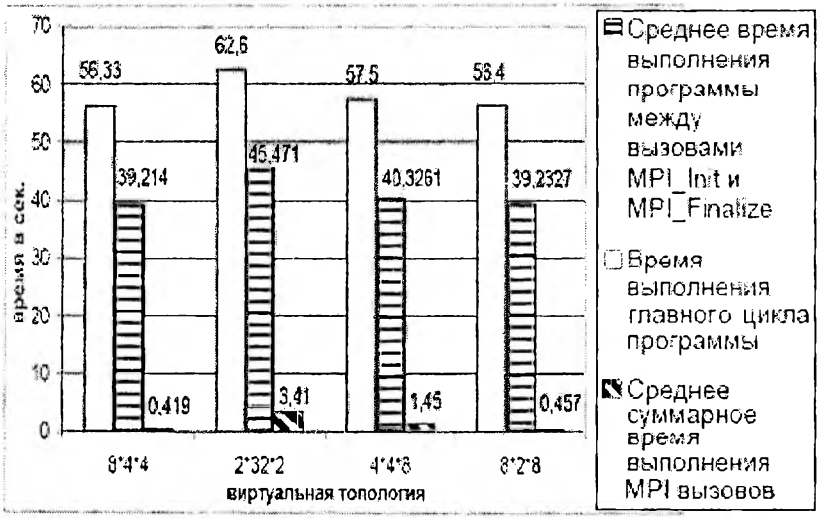


Рис. 5. Среднее время выполнения для различных виртуальных топологий в режиме SMP на 128 вычислительных узлах при распределении данных блоками.

Использование гибридных (DUAL и SMP) режимов выполнения при счете модельной задачи было эффективнее по времени, чем

использование режима VN, только при распределении данных полосами, которое требовало передачи очень больших по объему сообщений между MPI-процессами. При блочном распределении данных объем коммуникаций между MPI-процессами был существенно меньше и гибридные (DUAL, SMP) режимы выполнения проигрывали режиму VN, использующему только MPI для коммуникаций.

Для режима SMP и относительно небольшого размера партиции (128 выч. узлов) способ назначения процессов на процессоры еще не оказывает значительного влияния на время выполнения программы, только случайный mapping сильно отличается от других в худшую сторону. Суммарное время выполнения MPI-вызовов при случайном назначении процессов на процессоры примерно в 3.34 раза хуже, чем для «змеевидного» способа назначения (Рис.6).

В режиме DUAL (Рис.7) и VN (Рис.8) mapping оказывает уже гораздо большее влияние на время выполнения программы. При использовании стандартных способов назначения процессов на процессоры порядок координат X,Y,Z в перестановке при фиксированном положении T практически не влияет на время выполнения программы. Разница составляет примерно 0.3%, что сравнимо с различием времен при нескольких выполнениях программы с одним значением параметров. Само же положение координаты T в перестановке (первым или последним) достаточно существенно влияет на время выполнения программы: разница составляет примерно 3.5% в DUAL режиме и 13% в VN режиме.

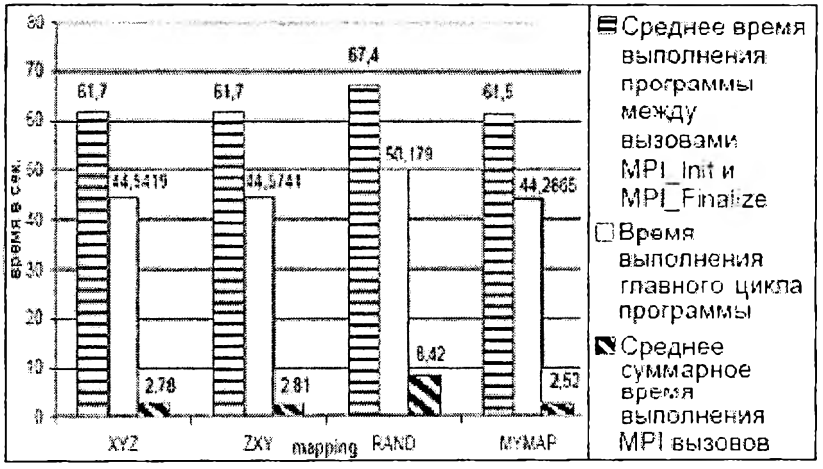


Рис. 6. Среднее время выполнения программы при различных способах назначения процессов на процессоры в режиме SMP на 128 вычислительных узлах при виртуальной топологии 1*1*128.

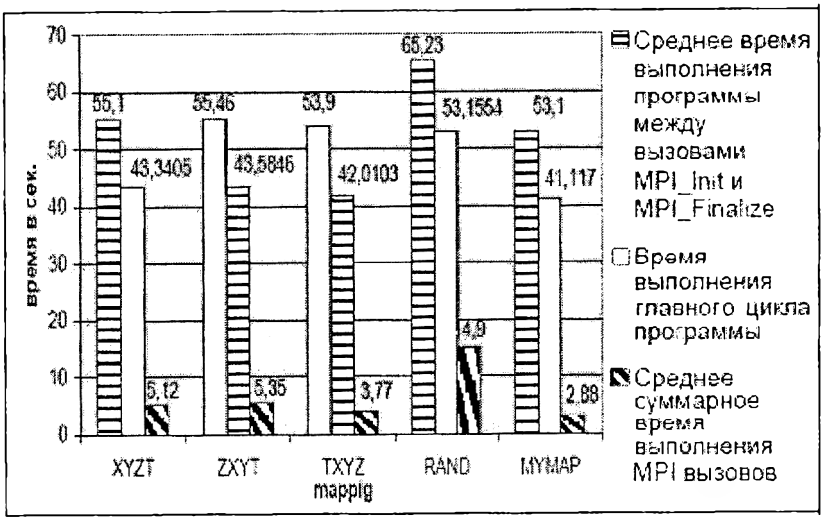


Рис. 7. Среднее время выполнения программы при различных способах назначения процессов на процессоры в режиме DUAL на 128 вычислительных узлах при виртуальной топологии 1*1*256.

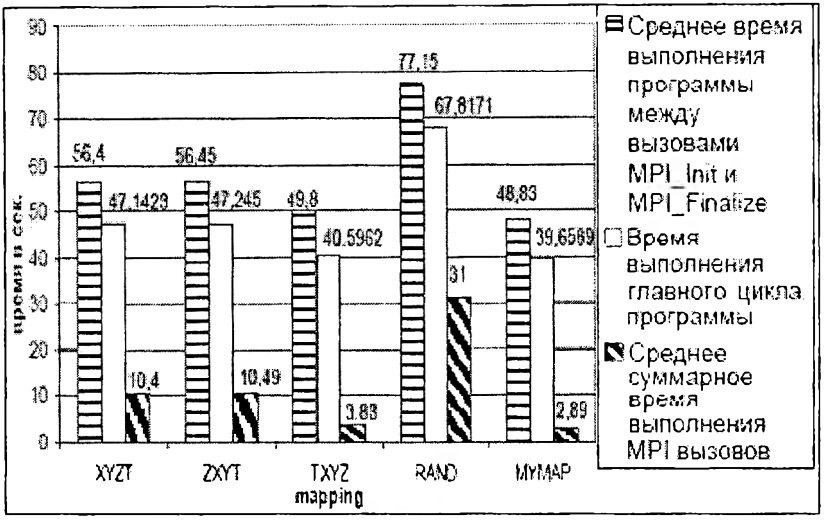


Рис. 8. Среднее время выполнения программы при различных способах назначения процессов на процессоры в режиме VN на 128 вычислительных узлах при виртуальной топологии 1*1*512.

Можно сделать вывод, что влияние способа назначения процессов на процессоры возрастает не только с числом

вычислительных узлов, но и с числом MPI процессов внутри вычислительного узла.

Эксперимент по исследованию масштабируемости проводился таким образом, чтобы не изменять размер передаваемых 1 узлом сообщений, то есть исследовалась масштабируемость по вычислениям, а не по коммуникациям (Рис. 9 - 10).

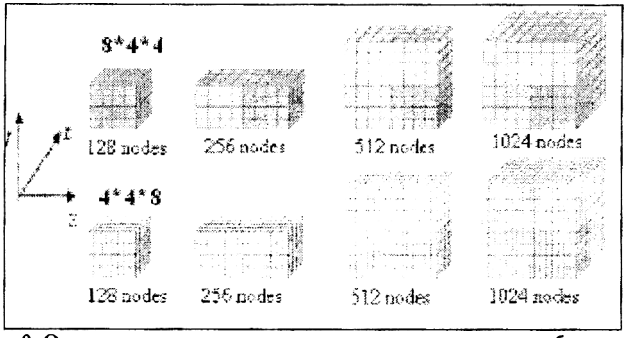


Рис. 9. Организация эксперимента по исследованию масштабируемости (направления увеличения размера задачи и партиции).

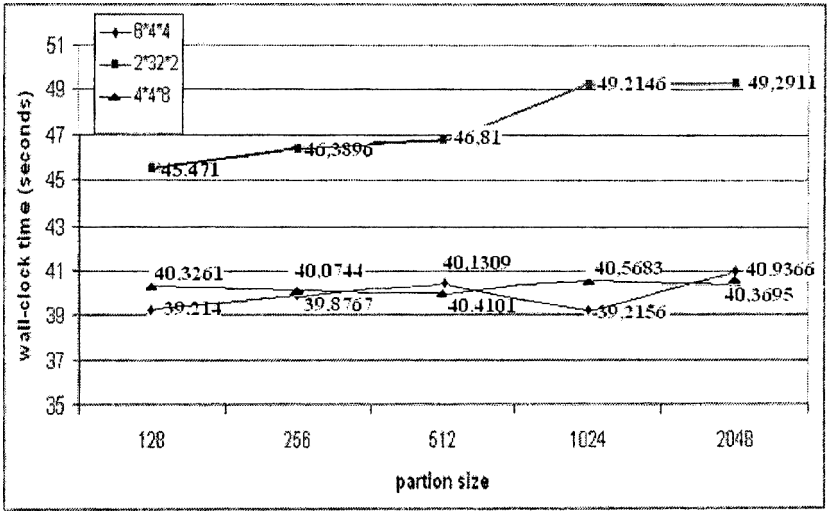


Рис. 10. Время выполнения главного цикла программы для различных виртуальных топологий и размеров партиции.

При неудачно выбранной виртуальной топологии (2*32*2) сильно возрастает длина пути передачи сообщений при увеличении размера партиции, потому приложение плохо масштабируется. Для виртуальных топологий (8*4*4, 4*4*8, 8*2*8) близких к физической

топологии системы наблюдается хорошая масштабируемость. Иногда с увеличением размера партиции происходило даже уменьшение времени выполнения задачи. Скорее всего, это связано с уменьшением нагрузки на сеть, создаваемой другими пользователями, и появлением новых путей передачи сообщений.

Заключение

Проведен обзор методов организации параллельных программ с использованием гибридного подхода. Обзор показывает актуальность и открытость проблемы.

Определена структура и методы организации данных в параллельной программе. Выполнена программная реализация, существенную роль в которой играет понятие "виртуальной топологии", которая позволяет совместить архитектуру системы и архитектуру задачи. Проведен вычислительный эксперимент, в результате которого определены параметры, влияющие на производительность, и выработаны некоторые рекомендации по выбору их значений.

Направления дальнейшего развития

- Исследование влияния размещения данных в памяти на время выполнения программы;
- Проведение исследования на других модельных задачах и тестовых пакетах, сравнение результатов для разных тестовых задач;
- Подключение снятия аппаратных характеристик и их анализ;
- Проведение аналогичного эксперимента на системе СКИФ-МГУ, сравнение и анализ результатов для двух систем.

Работа выполнена при поддержке грантов РФФИ 08-07-12081 и 08-07-00445-а.

Литература

1. TOP500 Supercomputing Sites. [HTML] (http://www.top500.org/).
2. Piotrowski M. Mixed Mode Programming on Clustered SMP Systems. 2006 [PDF] (http://www2.epcc.ed.ac.uk/msc/dissertations/dissertations-0506/2391976-9i-dissertation1.1.pdf).
3. Бахтин В.А., Коновалов Н.А., Крюков В.А., Поддерюгина Н.В., Сазанов Ю.Л. Разработка параллельных программ для решения больших вычислительных задач на SMP-кластерах. [PDF] (http:// lvk.cs.msu.su/files/mco2003/bahtin.pdf).

4. Makris I. Mixed Mode Programming on Clustered SMP Systems. 2005. [PDF] (http://www2.epcc.ed.ac.uk/msc/dissertations/dissertations-0405/6333284-9f-dissertation1.2.pdf).
5. He Y., Ding C. Hybrid OpenMP and MPI Programming and Tuning. 2004. [PPT] (http://www.nersc.gov/nusers/services/training/classes/NUG/Jun04/NUG2004_yhe_hybrid.ppt).
6. Nakajima K. OpenMP/MPI Hybrid vs. Flat MPI on the Earth Simulator: Parallel Iterative Solvers for Finite Element Method. 2003 [PDF] (http:// eofem.tokyo.rist.or.jp/report_common/GeoFEM03_007.pdf).
7. Cappello F., Etiemble D. MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks //ACM/IEEE conference on Supercomputing, Dallas, Texas, USA. 2000. [PDF] (http://www.sc2000.org/techpapr/papers/pap.pap214.pdf)
8. Buntinas D. MPI on Multicore Architectures using MPICH2. [PPT] (http://www.ncsa.uiuc.edu/UserInfo/Training/Workshops/Multicore/presentations/MPI_on_Multicore.ppt).
9. IBM System Blue Gene Solution: Blue Gene/P Application Development. 2008. [PDF] (http://www.redbooks.ibm.com/redbooks/pdfs/sg247179.pdf).
10. MPI: A Message-Passing Interface Standard // Message Passing Interface Forum. 2003. [HTML] (http://www.mpi-forum.org).
11. mpiP: Lightweight, Scalable MPI Profilingю [HTML] (http://mpip.sourceforge.net).

Князев Н.А., Сальников А.Н.

Управление динамическими приоритетами в планировании задач на многопроцессорных комплексах.

Предметная область

На сегодняшний день любой суперкомпьютер является многопроцессорной машиной.

А значит, возникают различные методы распределения задач на процессоры и вопрос эффективного использования вычислительных ресурсов. Системы, решающие такие задачи для сред распределенных вычислений, принято называть системы пакетной обработки (СПО). Благодаря им пользователь может помещать задания в общую для комплекса очередь, используя единый интерфейс для запуска, модификации, снятия и получения информация о заданиях.

СПО автоматически распределяет задания по узлам с учетом их загрузки, выполняет и доставляет результаты пользователю. Как правило, Массивно-параллельные системы имеют два уровня управления. На верхнем действует Планировщик, а на нижнем— Менеджер Ресурсов. Менеджер Ресурсов – программа позволяющая запускать конкретные задачи на конкретных узлах машины, т.е. управляет ресурсами компьютера. Планировщик – программа, которая в определенный момент времени определяет, какая из ожидающих выполнения задач будет передана Менеджеру Ресурсов.

В общем виде схема взаимодействия выглядит так:

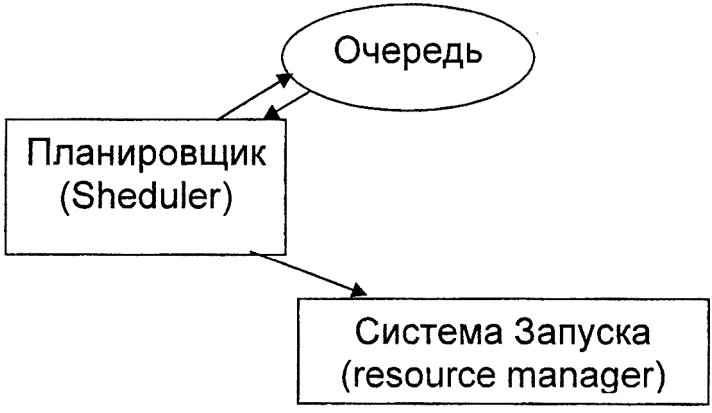


Рис 1.

Тематика данной статьи – этап планирования распределения заданий по ресурсам. На этом этапе на основе алгоритмов планирования вычисляется, какие задания и на каких ресурсах будут запущены. Объекты планирования – задания и ресурсы, задания запускаются пользователями, а ресурсы – это собственно вычислительные мощности системы.

Методы планирования

Современные методы планирования делятся на два класса:

- Методы, основанные на использовании очередей заданий.
- Методы, использующие для распределения ресурсов полноценный план на будущее или расписание.

Методы прохождения очередей

Традиционно такие методы применяются в кластерных системах управления, таких как PBS/Torque, LoadLeveler и других (В последней версии LoadLeveler есть возможность использования алгоритма backfill).

В простейшем случае задания запускаются одно за другим согласно их приоритетам, в этом случае загрузка ресурсов составляет примерно 50-80%[11]. В чистом виде не используется.

Существуют модификации. К примеру, модификация «FirstFit» допускает нарушение порядка очереди, позволяя низкоприоритетным заданиям занимать ресурсы, оставшиеся незагруженными.

Методы, использующие для распределения ресурсов полноценный план на будущее или расписание

Используют предварительное резервирование. В частности, метод обратного заполнения (Backfill). С их помощью автоматически обеспечивается накопление ресурсов и реализуется механизм предварительного резервирования ресурсов. Используются в Maui и последних версиях LoadLeveler.

Проблема приоритетов и обзор современных подходов к решению

Одной из актуальных проблем планирования задач является проблема: на чем основываться при определении, какую программу отдать системе запуска?

У большинства СПО при запуске программ есть возможность задавать приоритеты, или они проставляются автоматически.

Какие есть подходы к управлению приоритетами пользователей:

1. Каждый пользователь при запуске задачи проставляет приоритет данной задачи. В этом случае, многие пользователи, конечно, считают свои задачи самыми важными. Мы, с высокой вероятностью, получим, что все задачи запускаются с наивысшим приоритетом, и таким образом ранжирования задач по приоритетности теряет смысл.(пример MAUI Loadleveler).

2. Приоритеты связаны непосредственно с пользователями. В таком случае, существуют более приоритетные и менсе приоритетные пользователи и задачи, которые ставятся приоритетными пользователями выше. Но, в таком случае, возникает проблема, что у одного и того же пользователя могут быть более приоритетные и менее приоритетные задачи. И гораздо логичней и "честнее" вместо ненужных задач приоритетного пользователя пускать срочные задачи пользователя с меньшим приоритетом.

Используется также (к примеру, в MAUI) и смешанный подход, когда к приоритету пользователя прибавляется приоритет, который пользователь задает. В этом случае все недостатки первого подхода остаются, т.к. никаких ограничений на задаваемый приоритет в планировщике не предусмотрено.

Глобально проблема заключается в том, что пользователь знает приоритетность своих задач, но не может оценить приоритетность своей задачи по сравнению со всеми другими.

Обзор современных средств планирования и запуска задач пользователя

Первый шаг к использованию вычислительных ресурсов узлов в глобальной среде распределенных вычислений – это их локальная интеграция в многопроцессорный комплекс, который используется в режиме пакетной обработки заданий. Для этой цели и были разработаны программные продукты, называемые Системами управления пакетной обработкой (СПО). СПО возникли независимо от кластерных систем (систем, состоящих из множества компьютеров) и широко применяются во многих вычислительных центрах коллективной обработки. Известны более двадцати СПО, из которых наиболее популярны свободно распространяемые PBS(с расширениями Torque и планировщиком MAUI), SGE и Condor, а также коммерческие LoadLeveler и LSF.

1. PBS - Portable Batch System. [8]

PBS обеспечивает управление заданиями в пакетном режиме в среде компьютеров с ОС Unix. Используется устанавливаемый по

умолчанию планировщик FIFO (самый простой метод планирования), сконфигурированный для эксклюзивного выполнения одного счетного процесса на каждом из процессоров. PBS автоматически распределяет задания по свободным узлам заданной архитектуры. Никакого распараллеливания PBS не выполняет. Чтобы программа могла выполняться более чем на одном узле, она должна быть параллельной.

Существуют OpenPBS – свободно распространяемая версия и PBSPro – расширенная коммерческая версия.

Torque (Terascale Open-source Resource and QUEue Manager) – новая версия PBS, на основе OpenPBS. Несмотря на то, что в TORQUE имеется встроенная программа-планировщик (scheduler), pbs_sched, обычно он используется единственно как система управления заданиями (resource manager) совместно с планировщиком, делающим запросы к нему. В качестве планировщика чаще всего используют MAUI.

Система обладает рядом дополнительных усовершенствований:

-улучшена масштабируемость (работа в среде до 2500 узлов);

-повышена устойчивость к сбоям (внесены дополнительные проверки);

-усовершенствован интерфейс Планировщика с целью его обеспечения дополнительной и более точной информацией;

-усовершенствована система записей в файлах истории исполнения.

Как и в других СУПО планировщик PBS умеет выравнивать загрузку узлов (Loadleveling), опираясь на текущие полученные от ОС значения. Однако, кроме того, в PBS сделано важное движение в сторону распределения по текущему состоянию ресурсов.

2. Sun Grid Engine (SGE). [5] [6]

Семейство из нескольких различных вариантов СПО. Sun Grid Engine - свободно распространяемая версия, предназначенная для управления ресурсами одного проекта или подразделения. Основана на полной централизации обслуживания ресурсов и пользователей. Sun Grid Engine реализуется открытым сообществом разработчиков и спонсируется Sun Microsystems. SGE имеет лучший планировщик, чем старая OpenPBS, лучше относится к загрузке большим потоком малых задач. К сожалению, SGE в отличие от PBS не имеет развитого API для запуска и управления процессами, поэтому "тесная интеграция" ее с MPI приложениями обычно невозможна. Практически это значит, что PBS может чистить все MPI-процессы после аварийного завершения задачи, а SGE -- нет, в ней запущенные mpirun дочерние процессы остаются неубитыми.

Sun Grid Engine, Enterprise Edition (SGEEE) - коммерческая версия, предназначенная для управления ресурсами предприятий

(campus grid) и способная обслуживать несколько независимых проектов и групп пользователей. SGEEE включает модуль для определения политики разделения ресурсов между независимо работающими пользователями. Для каждого пользователя определяется квота от общего количества ресурсов, в соответствии с которой происходит их распределение между запущенными заданиями.

3. LoadLeveler [2]

Коммерческий программный продукт компании IBM, предназначенный для пакетной обработки последовательных и параллельных (многопроцессорных) заданий на кластерах из вычислительных серверов. Система обеспечивает средства для подготовки, запуска и слежения за заданиями в режиме пакетной обработки в гетерогенной сети компьютеров. Текущая версия дополнена рядом новых характеристик, таких как новые планировщики, механизм поддержки контрольных точек.

Последняя версия планировщика содержит также улучшенное восстановление буфера заданий. Эта функция используется для восстановления записей заданий из буфера, когда исходный управляющий процесс LoadL_schedd недоступен. Новая схема взаимодействия, разработанная для улучшения масштабируемости и производительности при запуске заданий, включает поддержку нескольких главных заданий в планировщике BACKFILL. Планировщик определяет недостаток необходимых ресурсов для выполнения задания и рассчитывает ближайшее время освобождения этих ресурсов.

Функция дополнительного планирования, которая позволяет TWS LoadLeveler дополнительно планировать выполнение нескольких шагов заданий, устанавливая минимальные ограничения на тип и число необходимых ресурсов.

4. MAUI [10]

Внешний планировщик, который может использоваться взамен штатных планировщиков для нескольких СПО: PBS, SGE, Loadleveler, LSF, Wiki. MAUI – продукт с открытым исходным кодом, который отличается большим набором режимов (политик) планирования и наличием механизма предварительного резервирования. Планировщик MAUI рассматривается отдельно, в связи с тем, что это единственная из свободно распространяемых систем планирования, которая позволяет автоматический запуск, избегая при этом неоправданного простоя ресурсов. Помимо того, в Maui есть ряд особенностей, Это:

- Алгоритмы обратного заполнения (backfill) и справедливого распределения ресурсов (fairshare) для повышения

эффективности системы и уменьшения времени ожидания в очереди. Эти алгоритмы используются также, в последних версиях Loadlevelera.

- Система автоматического определения приоритетов заданий, позволяющая давать ключевым пользователям преимущество при распределении ресурсов.
- Расширенный спектр статистики. С помощью Maui администратор может получать полную историческую и текущую информацию о заданиях, очередях, планировщике, состоянии системы и т.п.
- В Maui есть возможность моделирования работы СПО, с помощью которой можно проверить настройки планировщика "в деле". Это позволяет подобрать конфигурацию системы, наиболее полно отвечающую требованиям конкретной производственной обстановки.

MAUI имеет еще одно существенное отличие: обычные планировщики не имеют собственных командных интерфейсов. Maui же, полностью реализуя функции резервирования ресурсов для задач, располагает и соответствующим интерфейсом для внешнего, или, так называемого, административного резервирования

После выполнения некоторого конфигурируемого числа резервирований, начинает работу собственно backfill. Он выясняет, какие узлы и на какое время, начиная с текущего, свободны. После этого свободные узлы объединяются в окна. Суммарная "ширина" окон может оказаться больше количества свободных в данный момент узлов, т.к. некоторые узлы могут входить в несколько окон. Затем из всех окон выбирается одно (как правило, самое широкое). Среди всех оставшихся заданий выбирается наиболее точно удовлетворяющее этому "окну" задание и запускается. Если есть возможность, то запускается не одно задание. Так как при запуске заданий алгоритмом backfill учитываются сделанные резервирования, то соответствующие им задания не будут задержаны.

Таблица сравнения различных систем пакетной обработки

Параметр	PBS+Maui+Torque	SGE	LoadLeveler	МВС-1000
Открытость	Open Source	Отрытая и закрытая	Коммерческий продукт	Закрытая разработк
API	есть	есть	есть	есть
Приоритеты у задач	есть для работ	нет	есть	есть по времени
Политика запуска	Предварительное резервирование BackFill	Назначение задачам необходимых производите	Классы Работ и устройств - где что может запускаться	Задачи разделяются по типам
Разработчик	Maui и Torque - Clusterresources. PBS – Altair Engineering.	Sun	IBM	ИПМ им. Келдыша.

Рис 2.

Решение проблемы приоритетов

Как видно из предыдущих пунктов, на сегодняшний день по-прежнему очень актуальна разработка эффективного средства управления приоритетами.

Созданное средство должно позволять проставлять приоритеты внутри задач одного пользователя, в тоже время различные пользователи могли выставлять приоритет по отношению задач друг к другу. При этом не должно возникать гонки и постоянного выставления наивысшего приоритета.

Система должна быть разработана таким образом, что выставляемые приоритеты отражали реальное положение вещей.

В качестве решения предлагается создание невозобновляемых “фишек приоритета”.

Каждый пользователь обладает определенным набором фишек приоритета, которые он может назначать на задания при постановке в очередь.

Когда планировщик принимает решение о запуске задачи - предпочтение отдается задачам с большим приоритетом.

Таким образом, с одной стороны, каждый пользователь может определять приоритетность среди своих задач, с другой стороны, сохраняется “справедливое” распределение приоритетов среди всех пользователей.

Пользователи могут выбирать что важнее: запускать много мелких задач с малым приоритетом или одну с большим.

Со стороны администратора и владельцев среды распределенных вычислений появляется возможность продавать фишки на коммерческой основе или раздавать с учетом научных интересов.

Реализация предлагаемого решения

В качестве основы для реализации описанного решения был выбран планировщик MAUI.

В MAUI возможно назначать приоритет заданиям[3], т.о. возможно создать независимую программу, добавляющую в очередь MAUI задания, расставляя приоритеты, исходя из политики не возобновляемых фишек.

Как и любая СПО, разрабатываемая система разделена на клиентскую часть, работающую на front-end и серверную. Пользователь, минуя клиентскую часть MAUI, ставит задачу в очередь через клиент. Затем клиент, находящийся на машине front-end, отсылает задание и количество использованных фишек программе, находящейся на сервере, которая работает в фоновом режиме. Эта программа, получая задание, на основе израсходованных фишек создаёт задачу и приоритет , после чего передает её планировщику MAUI, с которым находится на одной машине. Доступ же к MAUI из front-end напрямую невозможен. Таким образом, не допускается запускать свои задачи, "вручную" проставляя приоритеты.

Схема работы создаваемой системы

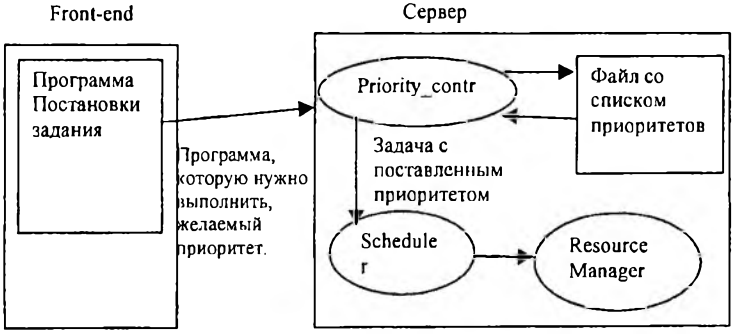


Рис.3.

Получая программу и приоритет от пользователя, работающий в фоновом режиме, обработчик проверяет в файле – есть ли достаточное количество фишек приоритета, и если да, то уменьшает их на

определенное количество и ставит задачу на выполнение планировщику MAUI с желаемым приоритетом.

Иначе берутся все неизрасходованные фишки.

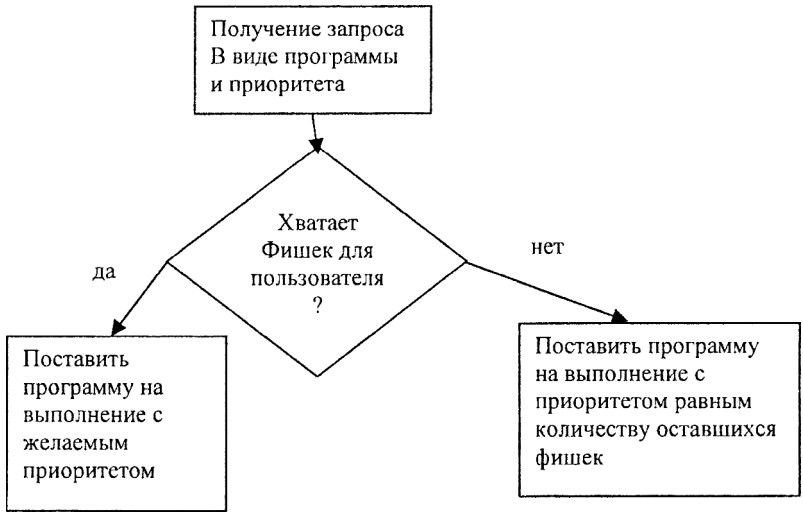


Рис.4.

Благодаря тому, что планировщик MAUI может работать совместно с большинством используемых СПО, предлагаемое решение также может использоваться в большинстве современных СПО. В частности использование MAUI в составе МВС-1000 через интерфейс Wiki было продемонстрировано в работе [1], аналогичным образом MAUI может использоваться и во многих других современных суперкомпьютерных комплексах.

Литература

1. А.В. Баранов, Д.М. Голинка. Исследование возможности использования планировщика Maui в составе СУПЗ МВС-1000.
2. Tivoli Workload Scheduler LoadLevelerUsing and Administering
3. Maui Administrator's Guide
4. Workload Management: SP and Other RS/6000 Servers Janakiraman Balasayee, Bruno Blanchard, Subramanian Kannan, Akihiko Tanishita (LoadLeveler)
5. Programming Interfaces Guide SunMicrosystems, Inc. (SGE)
6. System Interface Guide Sun Microsystems, Inc. (SGE)

7. Управление заданиями в распределенной среде и протокол резервирования ресурсов ИПМ им. М.В.Келдыша РАН Москва, 2002
8. TORQUE Administrator Manual
9. Руководство пользователя системы MBC-1000/RSC4. Авторы: А.В. Баранов О. Лацис С.В. Сажин М.Ю. Храмцов
10. Коваленко В.Н., Орлов А.В. (2002) "Управление заданиями в распределенной среде и протокол резервирования ресурсов" М. ИПМ им. М.В.Келдыша РАН
11. Quinn Snell, Mark Clement, David Jackson, Chad Gregory. The Performance Impact of Advance Reservation Meta-scheduling. Computer Science Department Brigham Young University Provo, Utah 84602-6576, 2000,
http://supercluster.org/research/papers/ipdps2000.pdf

Колесин М.С.

Исследование возможностей применения алгоритма колонии пчел для решения задачи нахождения оптимального мэппинга параллельной задачи на архитектуру BlueGene/P.

Работа выполнена при поддержке грантов РФФИ 08-07-0445 и 08-07-12081

Введение

Многие задачи оптимизации, зависящие от нескольких параметров не могут быть решены в полиномиальное время. Это рождает интерес к алгоритмам, которые могут находить приемлемые решения за подходящее время. К таким алгоритмам относятся т.н. *эволюционные* алгоритмы. Ключевое отличие этих алгоритмов от других заключается в том, что на каждом шаге они используют *популяцию* решений вместо одного решения. По ходу обработки популяции решений в течение одной итерации на выходе также получается популяция решений. В случае одного глобального оптимального значения, популяция должна сходится к нему. Когда же у задачи есть несколько оптимальных решений, эволюционный алгоритм может использоваться для заключения их всех в своей конечной популяции. К эволюционным алгоритмам относят *алгоритм муравьиной колонии* [2], *генетический алгоритм* [3], *алгоритм Particle Swarm Optimisation* (PSO) [4], *алгоритм колонии пчел* [5], а так же некоторые другие.

Общим для всех эволюционных алгоритмов является стратегия, по которой создаются варианты искомой популяции. Некоторые методы могут принимать новое решение в популяцию только если оно улучшает значение целевой функции, в то время как в других методах это не обязательно.

Алгоритм муравьиной колонии

Хорошим примером «нежадного» эволюционного алгоритма может служить алгоритм колонии муравьев.

Муравьи умеют перемещаться по сложной местности, находить пищу на большом расстоянии от муравейника и успешно возвращаться домой. Выделяя ферменты во время перемещения, муравьи изменяют окружающую среду, обеспечивают коммуникацию, а также отыскивают обратный путь в муравейник.

Они умеют находить самый оптимальный путь между муравейником и внешними точками. Чем больше муравьев используют

один и тот же путь, тем выше концентрация ферментов на этом пути. Чем ближе внешняя точка к муравейнику, тем больше раз к ней перемещались муравьи. Что касается более удаленной точки, то ее муравьи достигают реже, поэтому по дороге к ней они применяют более сильные ферменты. Чем выше концентрация ферментов на пути, тем предпочтительнее он для муравьев по сравнению с другими доступными. Так муравьиная "логика" позволяет выбирать более короткий путь между конечными точками.

Алгоритмы колонии муравьев интересны, поскольку отражают ряд специфических свойств, присущих самим муравьям. Муравьи легко вступают в сотрудничество и работают вместе для достижения общей цели. Алгоритмы муравьиной колонии работают так же, как муравьи. Это выражается в том, что смоделированные муравьи совместно решают проблему и помогают другим муравьям в дальнейшей оптимизации решения.

Генетический алгоритм

Для использования генетического алгоритма задача должна интерпретироваться таким образом, чтобы её решение могло быть представлено в виде вектора (*хромосома*). Случайным образом создаётся некоторое количество начальных векторов (*начальная популяция*). Они оцениваются с использованием функции приспособленности, в результате чего каждому вектору присваивается определённое значение (*приспособленность*), которое определяет вероятность выживания организма, представленного данным вектором. После этого с использованием полученных значений приспособленности выбираются вектора (*селекция*), допущенные к скрещиванию. К этим векторам применяются генетические операторы (в большинстве случаев скрещивание - crossover и мутация - mutation), создавая таким образом следующее поколение. Особи следующего поколения также оцениваются, затем производится селекция, применяются генетические операторы и т. д. Так моделируется эволюционный процесс, продолжающийся несколько жизненных циклов (*поколений*), пока не будет выполнен критерий останова алгоритма. Таким критерием может быть:

- нахождение глобального, либо субоптимального решения;
- исчерпание числа поколений, отпущенных на эволюцию;
- исчерпание времени, отпущенного на эволюцию.

Генетические алгоритмы служат, главным образом, для поиска решений в очень больших, сложных пространствах поиска.

Генетический алгоритм хорошо работает с данными большой размерности, но он недостаточно стабилен – в некоторых случаях не находится близкое к оптимальному решение.

Particle Swarm Optimisation

Алгоритм PSO предназначен для задач оптимизации, в которых решение может быть представлено в виде точки или поверхности в *n*-мерном пространстве. Этот алгоритм симулирует т.н. социальную оптимизацию.

Влияние общества позволяет людям избегать состояния когнитивного диссонанса, люди решают свои жизненные проблемы, обмениваясь убеждениями друг с другом, тем самым меняя свое поведение. Эти изменения могут быть представлены как движения людей друг к другу в социально-когнитивном пространстве.

В начале работы алгоритма формируется пространство индивидов, с начальными значениями целевой функции и «социальная сеть» - обеспечивающая взаимодействие между ними. Затем случайным образом формируется популяция индивидов, в качестве начальной популяции. Затем, итеративно, индивиды пытаются улучшить значение целевой функции, обмениваясь информацией о соседях друг с другом. В итоге популяция сходится к некоторой точке, которая и будет являться решением задачи оптимизации.

Частным случаем алгоритма PSO можно считать алгоритм колонии пчел. Он рассматривается в данной работе т.к. существенно проще и уже достаточно успешно применяется при решении разных задач.

Алгоритм колонии пчел

Пчелы в природе

Колония пчел может разлетаться на большие расстояния (до 14 км) и в разных направлениях с целью обнаружения наибольшего количества источников нектара. Одним из принципов, заложенных в развитии колонии, является то, что к источникам с большим количеством нектара направляется больше пчел. Процесс поиска начинается с того, что некоторое количество пчел-разведчиков отправляется по случайным направлениям в поисках нектара. После того как они возвращаются в улей, пчелы разведчики направляются в т.н. «танцевальный ярус» для осуществления «танца пчел». Этот танец является источником информации о местонахождении найденного источника для других пчел.

Пчелиный танец, использующийся для обмена данными, несет в себе следующую информацию:

- Местонахождение скопления цветов (угол между направлением на солнце и направление на источник нектара) – положение пчелы в пространстве.
- Расстояние от скопления цветов до улья – продолжительность танца.
- Количество нектара – частота танца.

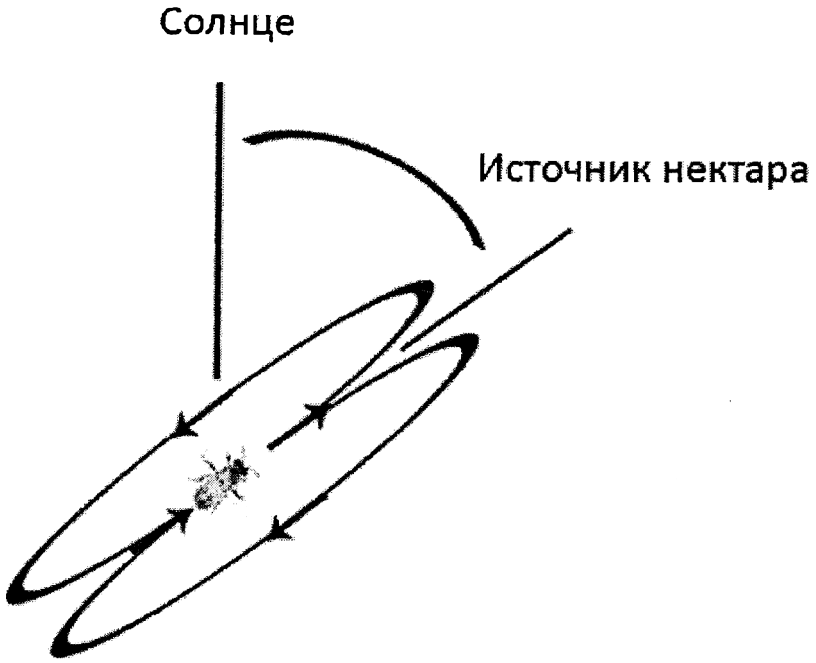


Рис. 1. Танец пчел.

После исполнения танца, пчела направляется обратно к источнику нектара с другими пчелами, ожидавшими в улье. Чем больше нектара в источнике – тем больше пчел туда полетят.

Схема работы алгоритма колонии пчел.

Для работы алгоритма необходимо установить несколько параметров, а именно:

- Число пчел-разведчиков (n).
- Число выбираемых для осуществления локального поиска источников (m).
- Число лучших источников из m выбранных (e).

- Число пчел, отправляемых к лучшим источникам (*nep*).
- Число пчел, отправляемых к оставшимся $m - e$ источникам (*nsp*).
- Критерий остановки работы алгоритма.

Можно описать схему работы алгоритма при помощи псевдокода:

```
1. Проинициализировать популяцию случайными решениями.
2. Рассчитать целевую функцию для каждого решения из
   популяции.
3. While (критерий остановки не выполнен) //Формируем новую
   популяцию.
    4. Выбрать решения для локального поиска.
    5. Направить пчел для поиска в окрестностях выбранных
       решений. Больше пчел для решений с большими значениями
       целевой функции.
    6. Выбрать решение с наилучшим значением целевой функции
       в каждой из окрестностей.
    7. Распределить оставшихся пчел для поиска новых
       случайных решений.
```

Ключевыми этапами работы алгоритма являются этапы 5 и 7. В некоторых реализациях алгоритма на 5-м шаге количество пчел, отправляемых к каждому из выбранных источников, является случайной величиной, распределение которой зависит от значения целевой функции в этом источнике. Как видно из приведенной выше схемы, на 6-м шаге выбирается только одно наилучшее найденное значение (лучшая пчела) из каждой окрестности. В природе этого ограничения не существует, оно введено, чтобы уменьшить кол-во проверяемых решений. Цикл 3-7 повторяется до тех пор, пока не выполнится критерий остановки алгоритма. В конце каждой итерации новая популяция будет составляться из двух частей – наилучших представителей уже исследуемых окрестностей и из отправленных по случайным направлениям разведчиков.

Результаты применения алгоритма колонии пчел для решения разных задач можно найти, например, в [6].

Математическая модель задачи мэппинга.

Модель мэппинга

Как известно, время работы параллельной программы складывается из времени, затраченного на вычисления, времени, затраченного на обмен информацией между процессами, а так же времени на синхронизацию процессов.

$$T = T_{выч} + T_{обм} + T_{синх} \quad (1)$$

То, каким образом процессы MPI-программы назначаются на физические узлы кластера в дальнейшем будем называть мэппингом. В данной работе будем полагать, что все узлы кластера обладают одинаковыми вычислительными ресурсами. В этом случае мэппинг не будет влиять на первое слагаемое из правой части (1). Очевидно, что мэппинг может существенно влиять на время, затрачиваемое на обмен сообщениями и синхронизацию между MPI-процессами. В рамках данной работы будем рассматривать влияние мэппинга только на время обменов $T_{обм}$.

Рассмотрим следующую математическую модель.

Пусть известно N_p - число выделенных логических процессоров (далее просто процессоров) кластера, и N_t - число MPI-процессов, необходимых для решения задачи. Для простоты предположим, что $N_p = N_t = N$.

Рассмотрим две матрицы $C_{N\times N}$ и $S_{N\times N}$, где c_{ij} - это объем информации, которым обмениваются MPI-процессы с номерами i и j во время выполнения задачи, а s_{ij} - «стоимость» обмена информацией между процессорами с номерами i и j. Матрица C определяется только MPI-приложением, а матрица S – только характеристиками кластера, такими, как топология, используемая среда передачи данных между узлами и т.п.

Таким образом, задачу нахождения оптимального мэппинга можно сформулировать как минимизацию значения функции

$$E[f] = \sum_{i,j=1}^{N} C(i,j)S(f(i),f(j)), \quad (2)$$

где $f(i)$, $i = \overline{1,N}$ - функция мэппинга, ставящая в соответствие номеру MPI-процесса номер процессора, на котором этот процесс будет выполняться. Данная функция должна ставить в соответствие разным процессам разные процессоры, т.к. в противном случае некоторые процессоры будут простаивать во время выполнения задачи, что не

является эффективным использованием кластера. Это дает возможность рассматривать функцию мэппинга как перестановку из N чисел от 1 до N, где на i-м месте стоит номер процессора, на котором будет запускаться i-й MPI-процесс.

Итак, решением задачи поиска оптимального мэппинга параллельной задачи на кластер будет нахождение такой перестановки $(p_1 \quad p_2 \quad \dots \quad p_N)$, при которой значение функции

$$E[f] = \sum_{i,j=1}^{N} C(i,j)S(p_i, p_j)$$

с заданными для данной задачи и кластера матрицами C и S будет минимальным.

Исходя из постановки задачи, получаем пространство поиска решений размером $N!$.

Применение алгоритма колонии пчел

Рассмотрим некоторые особенности реализации пчелиного алгоритма для решения задачи мэппинга.

Случайные решения.

Для генерации случайных перестановок в работе используется модифицированный алгоритм Фишера-Йейтса, который заключается в следующем:

1. Пусть $n = N$, $p_1 = 1, p_2 = 2, \dots, p_n = n$.
2. Выбираем случайное число $k \in \overline{1, n-1}$
3. Меняем в исходной перестановке местами p_k и p_n.
4. Уменьшаем n на единицу.
5. Повторяем шаги со 2го по 4й, пока n больше 2.

Поиск в окрестности решения.

Для поиска в окрестности выбранной перестановки, будем менять местами два случайных элемента перестановки. Например, при $N = 5$, в окрестности перестановки

$$(1,2,3,4,5)$$

будут находиться:

$$(2,1,3,4,5), (3,2,1,5,4), (4,2,3,1,5), \dots,$$
$$(1,3,2,4,5), (1,4,3,2,4), (1,5,3,4,2), \dots,$$
$$\vdots$$

Результаты

Топология кластера

При тестировании работы алгоритма моделировалась следующая ситуация.

Предположим, что кластер представляет из себя K однородных четырехядерных процессоров, и обладает топологией трехмерного куба, в котором у каждого из процессоров может быть не более 6 напрямую соединенных с ним соседей.

Тогда имеем $N = 4K$ логических процессоров, на которых независимо будут запускаться MPI-процессы.

В таком случае, матрица S при $N = 256$ будет иметь вид:

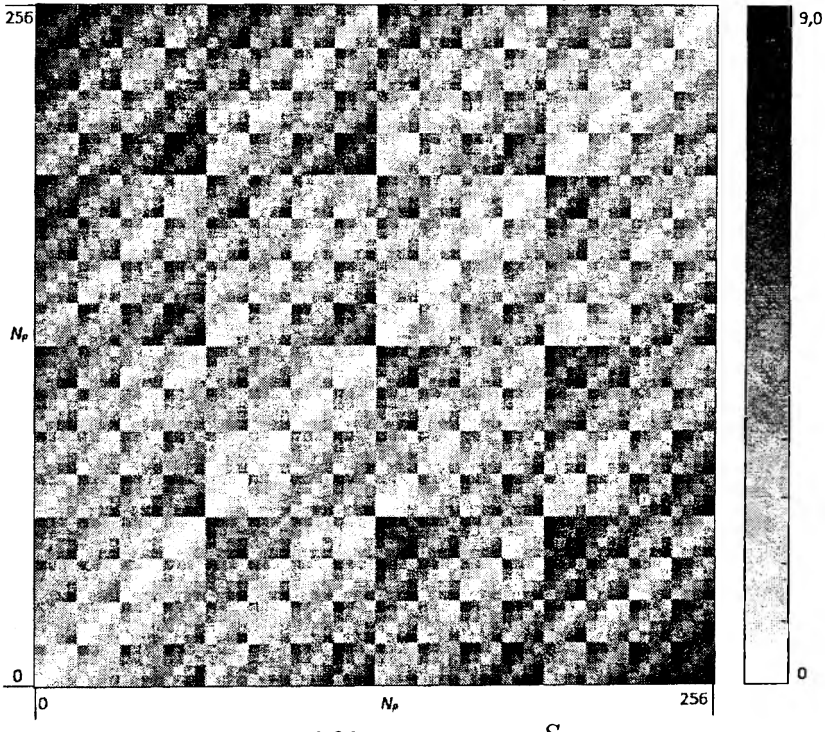


Рис. 2 Общий вид матрицы S

Такая топология схожа топологии кластера Bluegene/P в режиме SMP и является достаточно точной для тестирования масштабируемости и эффективности работы алгоритма.

Тестовые задачи

При тестировании было использованы коммуникационные матрицы двух типов MPI-приложений: передача сообщения по цепочке и реализация функции AllToAll.

Передача по цепочке

При передаче по цепочке каждый процесс с номером i обменивается фиксированным объемом информации с процессами $i-1$ и $i+1$. В этом случае матрица C имеет вид:

$$\begin{matrix} 0 & 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & 0 & \cdots & 1 & 0 \end{matrix}$$

N

Рис. 3 Общий вид матрицы C

Оптимальным мэппингом для такой задачи будет прямой мэппинг

$$p_1 = 1, p_2 = 2, ..., p_N = N.$$

Для выяснения эффективности работы алгоритма, была вычислена целевая функция при прямом (оптимальном) мэппинге, а также целевая функция для лучшего найденного мэппинга на каждой итерации алгоритма.

Получены результаты работы алгоритма для $N = 128$ и $N = 256$ с параметрами $n = 100$, $m = 40$, $e = 20$, $bfbes = 100$, $bfos = 50$, $bsc = 500$, $Iterations = 200$:

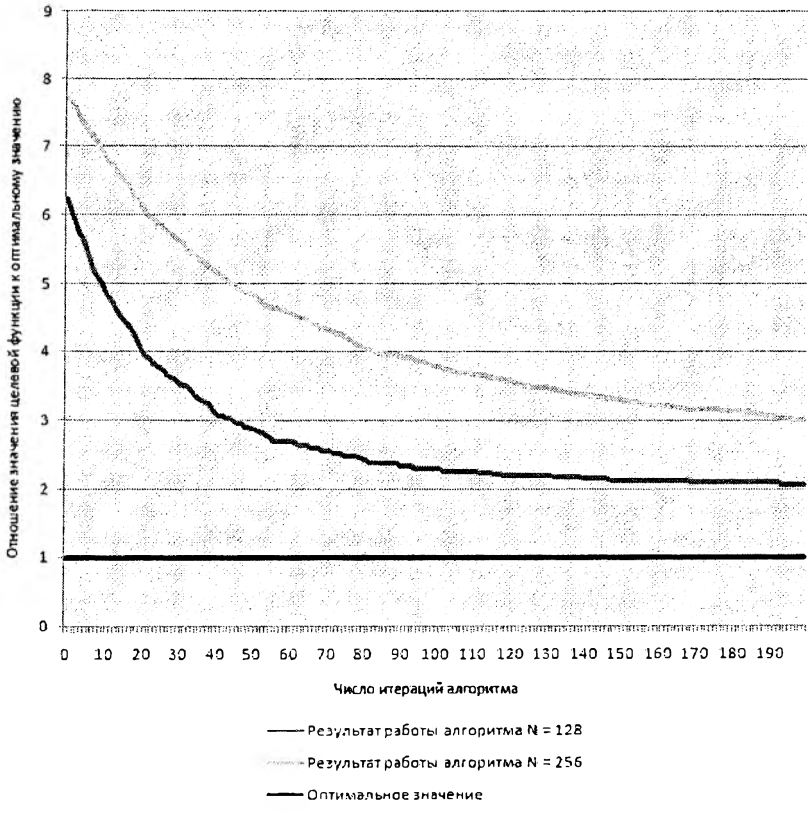


Рис. 4. Сравнение результатов при $N = 128$ и $N = 256$

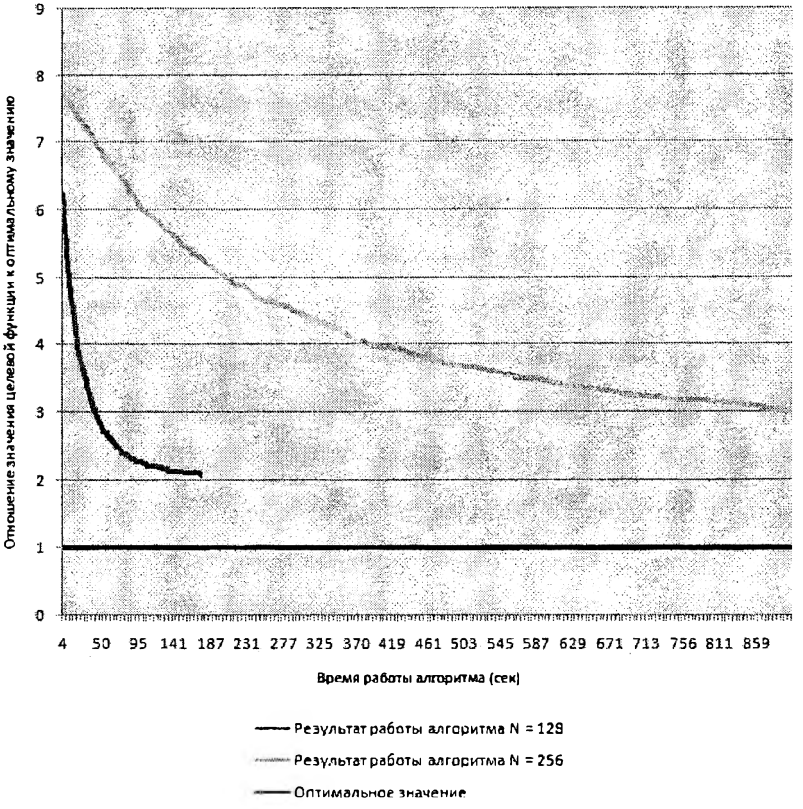


Рис. 5. Сравнение результатов работы алгоритма при $N = 128$ и $N = 256$ (с учетом времени работы алгоритма)

Функция AllToAll

При простейшей реализации функции AllToAll коммуникационная матрица имеет следующий вид (в данном примере $N = 256$):

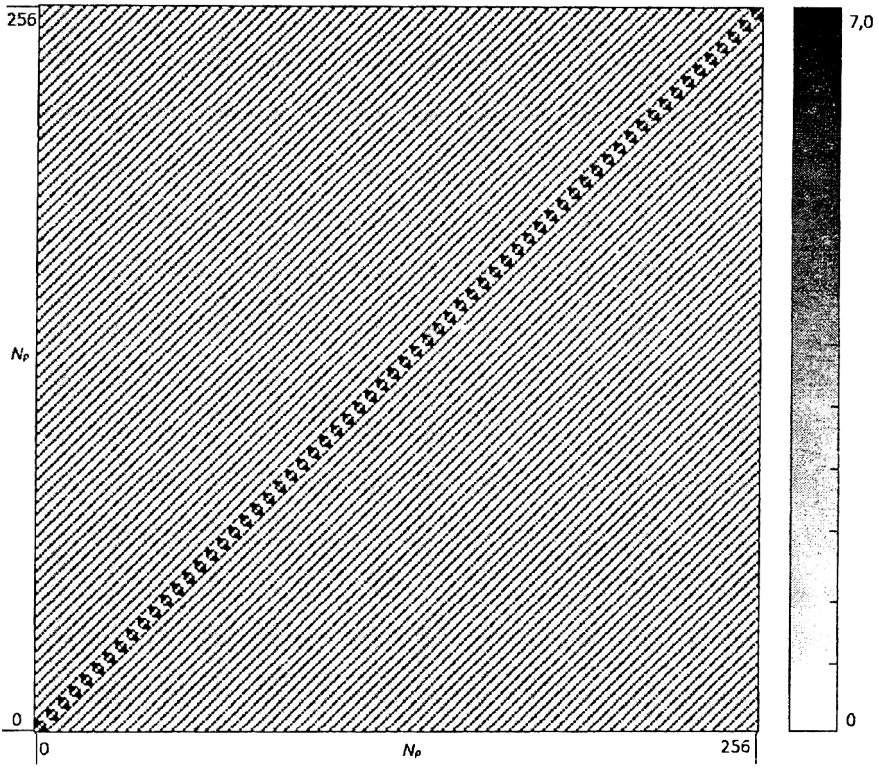


Рис. 6. Вид коммуникационной матрицы для функции AllToAll

Получены результаты работы алгоритма для $N = 256$ с параметрами $n = 100$, $m = 40$, $e = 20$, $bfbes = 100$, $bfos = 50$, $bsc = 500$, $Iterations = 200$.

График изменения значения целевой функции представлен на рис. 7. Видно, что за 200 итераций алгоритма значение целевой функции уменьшается более чем на 25 процентов.

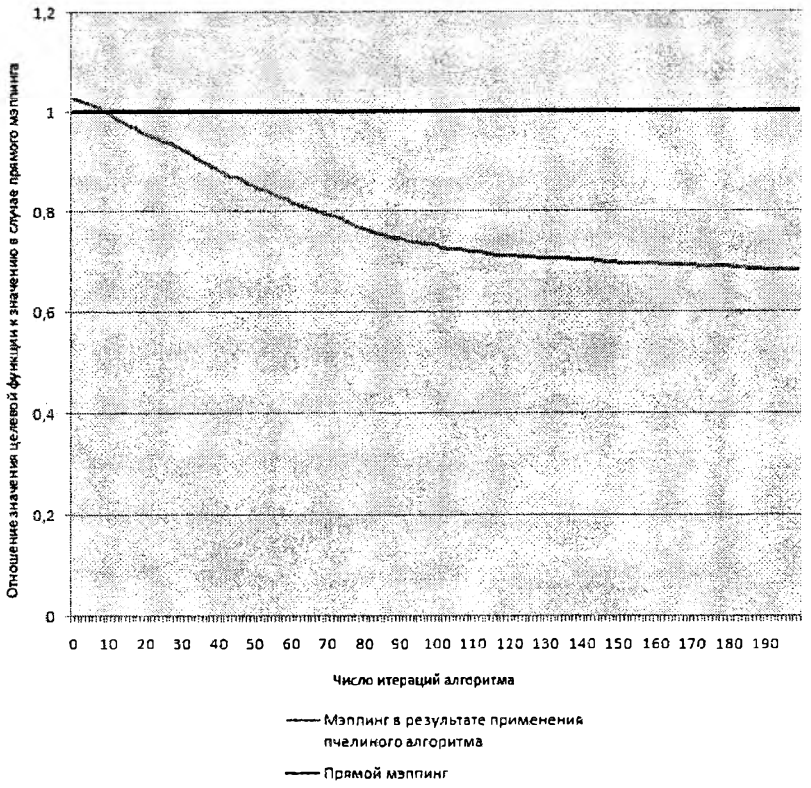


Рис. 7 Изменение значения целевой функции ($N = 256$) в случае коммуникационной матрицы AlltoAll

Более наглядными результаты делает следующая интерпретация. В качестве карты распределения обменов сообщениями можно рассматривать матрицу, которая получается в результате поэлементного перемножения матриц S и C. Таким образом, сумма элементов полученной матрицы будет являться ничем иным, как значением целевой функции мэппинга. На рис. 8 и рис. 9 представлены карты обменов сообщениями для прямого мэппинга функции AlltoAll и для мэппинга, полученного пчелиным алгоритмом. Из рисунков видно, что мэппинг, получаемый алгоритмом, «подстраивается» под структуру матрицы S, в результате чего и получаем существенное улучшение значения целевой функции.

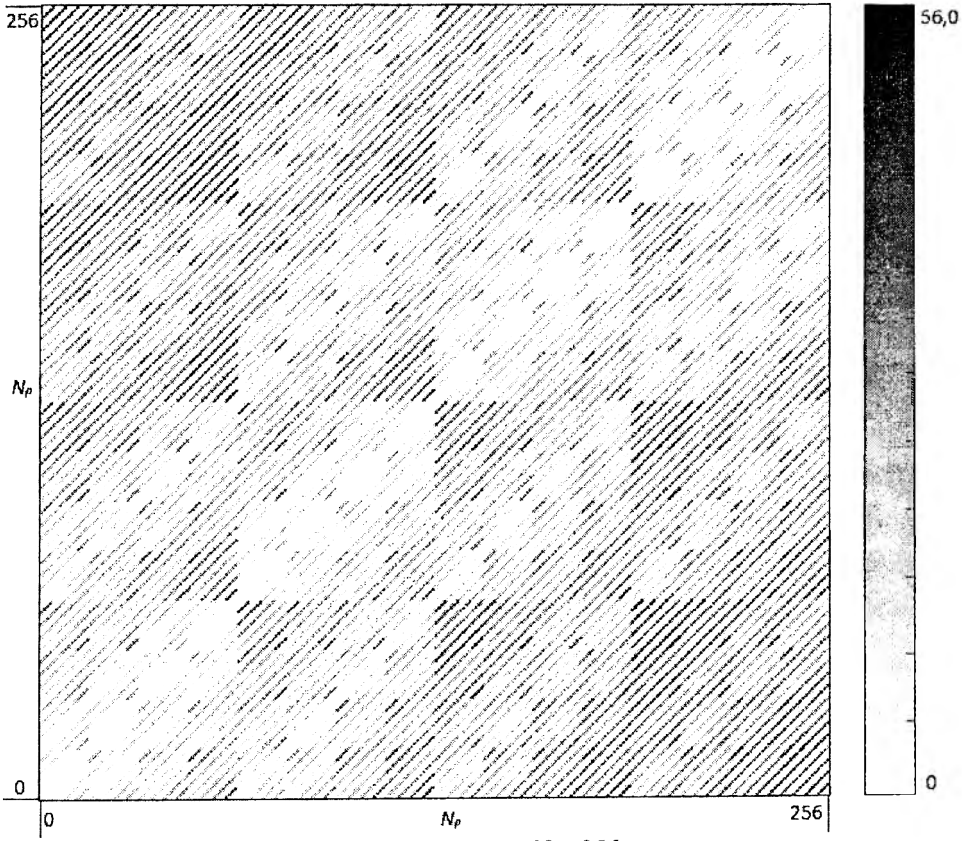


Рис. 8. Прямой мэппинг AllToAll ($N = 256$)
Карта распределения обменов сообщениями.

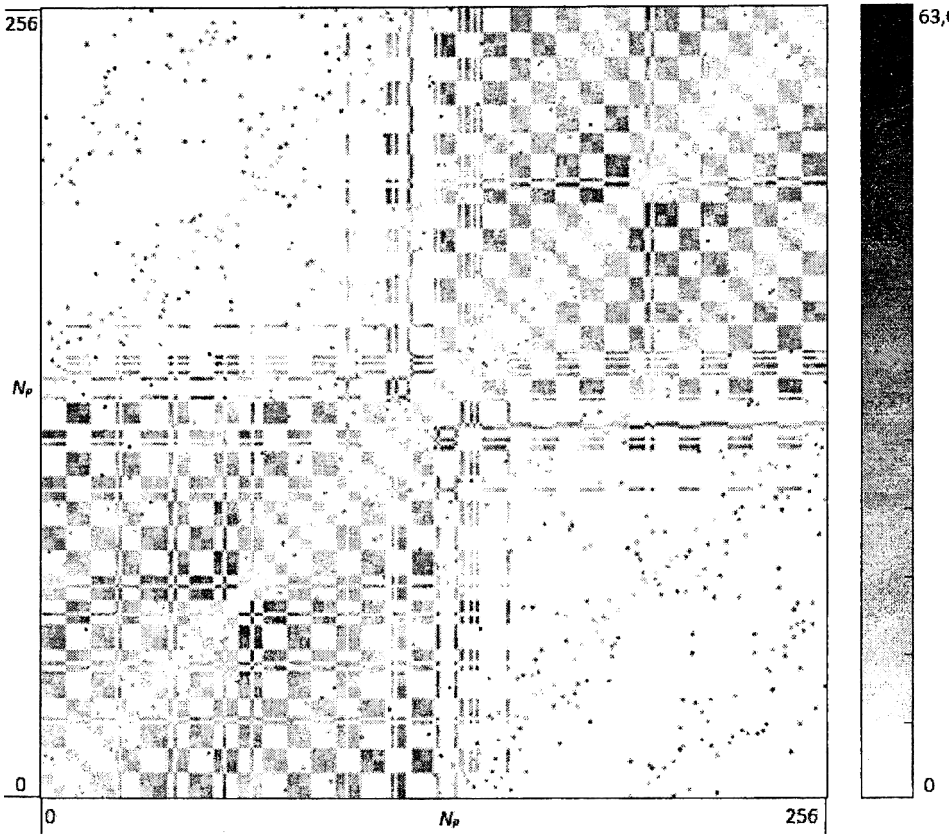


Рис. 9. Мэппинг AllToAll, полученный пчелиным алгоритмом ($N = 256$)
Карта распределения обменов сообщениями.

Заключение

По результатам тестирования работы алгоритма на примере функции AllToAll видно, что алгоритм способен подстраиваться под топологию кластера и показывать неплохие результаты с точки зрения уменьшения значения целевой функции.

Однако, как видно по рис. 5, при увеличении числа процессоров, используемых задачей, со 128 до 256 время работы алгоритма возрастает в 4 раза. Это происходит из-за двукратного увеличения размеров матриц, и соответствующего увеличения времени на расчет целевых функций популяции. Эту проблему можно решить, реализовав параллельную версию алгоритма. Такой подход будет работать, т.к. можно производить независимое разделение областей поиска между параллельными процессами, что существенно уменьшит время выполнения алгоритма. Последовательная же версия алгоритма неприемлемо медленно работает уже при $N = 512$, что осложняет использование этого способа нахождения оптимального мэппинга в реальных задачах.

Также возможно повысить эффективность алгоритма за счет более правильного (с точки зрения топологии кластера) формирования окрестностей решений. В текущем варианте производится лишь случайная перестановка двух элементов решения для формирования «соседа».

Еще одним направлением для дальнейших исследований является подбор оптимальных параметров алгоритма для повышения его эффективности и/или уменьшения времени работы.

В качестве вывода, сделанного по выполнении данной работы можно отметить, что алгоритм колонии пчел может быть эффективно применен к решению задачи нахождения оптимального мэппинга. Классический последовательный вариант работает достаточно эффективно при небольшом количестве процессоров (а значит небольшом пространстве поиска решений). Параллельная же реализация алгоритма представляется более перспективной.

Литература

1. http://www.bees-algorithm.com/
2. *Dorigo M and Stützle T. Ant Colony Optimization. MIT Press,* Cambridge, 2004
3. Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Reading: Addison-Wesley Longman, 1989.
4. Eberhart, R., Y. Shi, and J. Kennedy, Swarm Intelligence. Morgan Kaufmann, San Francisco, 2001
5. D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi. The Bees Algorithm – A Novel Tool for Complex Optimisation Problems. Manufacturing Engineering Centre, Cardiff University, Cardiff CF24 3AA, UK, 2006.
6. D.T.Pham, E. Koç, A. Ghanbarzadeh, S. Otri. Optimisation of the Weights of Multi-Layered Perceptrons Using the Bees Algorithm. Proceedings of 5th Inernational Symposium on Intelligent Manufacturing Systems, May 29-31, 2006, p. 38-46.
7. R. Sedgewick. Permutation Generation Methods. Computing Surveys, june 1977, v. 9, n. 2.
8. Task Mapping Problem, Method and Results, Irina Fedulova, IBM, April 2009.
9. Optimizing task layout on the Blue Gene/L supercomputer, G. Bhanot, A. Gara, P. Heidelberger, E. Lawless, J. C. Sexton, R. Walkup, IBM J. RES. & DEV. VOL. 49 NO. 2/3 MARCH/MAY 2005.

Раздел III
К теории формальных языков и обработки текстовой информации

Вылиток А.А., Сутырин П.Г., Харченков С.Л., Юдочев Д.В.

О сопряжениях графовых описаний формальных языков

Введение

Среди конечных описаний формальных языков можно выделить распознающие (например, конечные и магазинные автоматы) и порождающие (например, формальные грамматики). В настоящей работе предлагаются графовые описания формальных языков, которые сочетают в себе свойства распознающих и порождающих описаний и позволяют единообразно рассматривать регулярные и контекстно-свободные языки. Для них предложен алгоритм построения сопряжений, связанный с задачей проверки равенства формальных языков (т.е. эквивалентности соответствующих описаний).

Вылитком А.А. разработан аппарат сопряжений, Сутыриным П.Г. построено обобщение D-графов как графовых описаний, Харченковым С.Л. предложен алгоритм построения сопряжений, Юдочевым Д.В. построено обобщение конечных автоматов как графовых описаний и для них доказана теорема о развитии.

1. Основные определения

Напомним некоторые понятия теории формальных языков, используемые далее в тексте. Для конечного алфавита (*терминальных*) символов Σ (например, $\Sigma = \{a,b,c\}$) через Σ^* будем обозначать множество всех цепочек над алфавитом Σ, включая пустую цепочку ε. Иногда будем различать обозначения пустых цепочек над разными алфавитами. Множество всех непустых цепочек над алфавитом Σ будем обозначать через Σ^+. Формальный язык L — это подмножество множества Σ^*, $L \subseteq \Sigma^*$.

Пусть $\Sigma_($ и $\Sigma_)$ — непересекающиеся алфавиты (*скобочных*) символов, соответственно *открывающих* и *закрывающих* (будем называть их просто *скобками*). Введем обозначение: $\Delta = \Sigma_(\cup \Sigma_)$. Пустую цепочку над алфавитом Δ будем обозначать Λ. Назовем непустое

множество $P \subseteq \Sigma_(\times \Sigma_)$ *P D-множеством*. D-множество представляет собой множество допустимых пар скобочных символов, причем одной и той же открывающей скобке может соответствовать более чем одна закрывающая (например, $P = \{(a,b), (a,c)\}$) и наоборот. Язык $L_P \subseteq \Delta^*$, порождаемый контекстно-свободной грамматикой $G_P = (\Delta, \{S\}, \{S \to \Lambda\} \cup \{S \to aSbS, (a, b) \in P\}, S)$, будем называть *D-языком* L_P (*над D-множеством* P). Элементы $x \in L_P$ будем называть *скобочными системами*.

2. Графовые описания формальных языков

Далее даются определения конечного автомата и *D*-графа. Используется форма определения конечного автомата, приближенная к определению *D*-графа, что позволит нам рассматривать эти два класса объектов единообразно. Итак, *графовое описание (ГО)* — это конечный автомат или *D*-граф.

Конечный автомат — это шестёрка $A = (\Sigma, V, E, \lambda, P_0, F)$, где Σ — конечный алфавит пометок дуг; V — конечное множество вершин; E — конечное множество дуг; $\lambda : E \to V \times (\Sigma \cup \{\varepsilon\}) \times V$ — функция положения дуги; P_0 — начальная вершина, $P_0 \in V$; F — множество заключительных вершин.

Перейдем к определению D-графа. Пусть имеются непересекающиеся алфавиты скобок $\Sigma_($ и $\Sigma_)$, и *D*-множество P. Введем обозначения:

$Left(P) = \{a \in \Sigma_(\mid \exists\, b \in \Sigma_) : (a,b) \in P\}$;

$Right(P) = \{b \in \Sigma_) \mid \exists\, a \in \Sigma_(: (a,b) \in P\}$;

D-графом будем называть шестерку $D = (\Sigma, V, P, \lambda, P_s, F)$, где Σ — непустой алфавит терминальных символов, $V = V(D)$ — непустое множество *вершин*, P — *D*-множество. Элемент π множества $E = E(D) = Left(P) \cup Right(P)$ будем называть *дугой*. *D*-язык, отвечающий *D*-множеству P, обозначим через $L_P(D)$ и отметим, что $L_P(D) \subseteq E(D)^*$.

Дальнейшие определения будем вводить для графового описания *G*. Некоторые понятия будут определены по-разному, в зависимости от того, является ли графовое описание *G* конечным автоматом или *D*-графом.

Элемент $\lambda : E(D) \to V \times (\Sigma \cup \{\varepsilon\}) \times V$ — *функция положения* (дуги в графовом описании *G*); элементы тройки

$(P,a,Q) = \lambda(\pi)$ для некоторой дуги π, будем называть соответственно *начальной вершиной*, *пометкой* и *конечной вершиной* этой дуги, и по отдельности обозначать соответственно как $beg(\pi) = P$, $\omega(\pi) = a$, $end(\pi) = Q$. $P_s \in V$ — *начальная вершина*, $F \in V$ — множество *заключительных вершин*.

Пара $(V(G), E(G))$ образует ориентированный мультиграф с помеченными дугами (см., например, [1]). В соответствии с этим *путь* в ГО G определим как цепочку $T = P_0 \pi_1 P_1 \pi_2 P_2 \ldots \pi_n P_n$, где $n \geq 0$; для $i = 0,\ldots,n$ $P_i \in V(G)$; для $i = 1,\ldots,n$ $\pi_i \in E(G)$, $beg(\pi_i) = P_{i-1}$, $end(\pi_i) = P_i$. Длину n пути T обозначим через $| T |$, путь $T = P_0$ нулевой длины будем называть *пустым*. *Начальную* P_0 и *конечную* P_n вершины пути T будем обозначать соответственно через $beg(T)$ и $end(T)$. Множество всех путей в ГО G обозначим через $Paths(G)$.

Понятие *пометки пути* в ГО G определим рекурсивно: пометка пустого (или тривиального) пути пуста, а для пути $T\pi$, в котором $\pi \in E(G)$ и , следовательно, $\lambda(\pi) = (P,a,Q)$ для некоторых $P,Q \in V(G)$ и $a \in \Sigma \cup \{\varepsilon\}$, пометка есть $\omega(T\pi) = \omega(T)a$.

Путь T' будем называть *фрагментом* пути T, если существует представление $T = T_1 T' T_2$, где T_1, T_2 — пути.

Предложением ГО G будем называть путь $T \in Paths(G)$, такой, что $beg(T) = P_s$, $end(T) \in F$, и $T \in \mathcal{L}_\mu(G)$ в случае, если G — D-граф. Множество всех предложений ГО G обозначим через $Sentences(G)$. Путь T' будем называть *маршрутом*, если он является фрагментом некоторого предложения. Если G — D-граф, то его маршрут T' будем называть *нейтральным*, если $T' \in \mathcal{L}_p$. Если же G — конечный автомат, то все его маршруты будем считать нейтральными. *Циклом* в ГО G будем называть такой непустой маршрут T, что $beg(T) = end(T)$.

Языком $L(G) \subseteq \Sigma^*$, задаваемым ГО G, будем называть множество всех пометок его предложений: $L(G) = \{x \in \Sigma^* \mid x = \omega(T), T \in Sentences(G)\}$. Два ГО G_1 и G_2 будем называть *эквивалентными* (и обозначать как $G_1 \sim G_2$), если $L(G_1) = L(G_2)$.

Пусть маршрут $T = T_0\pi_1 T_1 T_2 T_3 \pi_3 T_4$ D-графа G, таков, что π_1 и π_3 являются дугами и маршруты $T_2, T_1 T_2 T_3, \pi_1 T_1 T_2 T_3 \pi_3 \in \mathcal{L}_p$. Тогда назовем тройку $(\pi_1 T_1, T_2, T_3\pi_3)$ *гнездом* (маршрута T). Иногда будем называть гнездом и маршрут $\pi_1 T_1 T_2 T_3 \pi_3$. Будем говорить, что участки $\pi_1 T_1$, $T_3\pi_3$ *парны* (друг другу), или *образуют гнездо* в маршруте T, будем называть

их *открывающим* и *закрывающим* (в этом гнезде) соответственно.

Конечный автомат G назовем *детерминированным*, если для любой пары дуг π_1 и π_2, таких что $beg(\pi_1) = beg(\pi_2)$, выполняются условия:
1) $w(\pi_1) \neq w(\pi_2)$.
2) $w(\pi_1) \neq \varepsilon$, $w(\pi_2) \neq \varepsilon$.

D-граф G назовем *детерминированным*, если для любой пары дуг π_1 и π_2, таких что $beg(\pi_1) = beg(\pi_2)$, $w(\pi_1) = w(\pi_2)$ или $w(\pi_1) = \varepsilon$ выполняются условия:

1) π_1 и π_2 - закрывающие скобки;
2) $left(\pi_1) \cap left(\pi_2) = \varnothing$.

Пусть T — маршрут ГО G и $w \geq 0$. Пусть максимальное число циклов $T_1,\ldots,T_m$ таких, что цикл $T_1 \ldots T_m$ является участком маршрута T, равно w. Тогда будем говорить, что *циклическая ширина* маршрута T равна w, и обозначать: $CycleWidth(T) = w$.

Пусть T — маршрут D-графа G и $d \geq 0$. Пусть, аналогично, для каждого числа $m \geq 0$ из некоторого конечного множества $\{m\}$ найдутся циклы $T_{11},\ldots,T_{1m}$, T_{21}, $T_{31},\ldots,T_{3m}$, для которых $T_{2(i+1)} = T_{1i}T_{2i}T_{3i}$, $1 \leq i \leq m$, есть гнездо маршрута T, образованное циклами T_{1i} и T_{3i}. Если $\max\{m\} = d$, то будем говорить, что циклическая глубина маршрута T равна d, и обозначать: $CycleDepth(T) = d$. Случай $d = 0$ означает, что в маршруте нет ни одного гнезда, образованного парными циклами. Гнезда, образованные парными циклами будем называть циклическими гнездами. Для любого маршрута T конечного автомата G положим $CycleDepth(T) = max\{CycleWidth(T),1\}$.

Пусть T — маршрут КА A, w — неотрицательное число. (w)-*каноном* автомата A назовем его маршрут T, такой, что $CycleWidth(T) \leq w$.

Пусть T — маршрут D-графа D, w,d — неотрицательные числа. (w,d)-каноном D-графа D назовем его маршрут T, такой, что $CycleWidth(T) \leq w$ и $CycleDepth(T) \leq d$.

Для неотрицательного целого числа w назовем w - *ядром* ГО G: для КА G — множество всех его (w)-канонов, для D-графа G — множество всех его (w,w)-канонов. Будем обозначать w-ядро ГО G следующим образом: $Core(G,w)$.

В [2] вводятся понятия развития маршрутов D-графа D, предка

и родословной маршрута, и показывается, что всякий маршрут D-графа D может быть получен развитием некоторого маршрута (называемого его предком) из (2,1)-ядра D-графа. Мы же введем эти понятия для конечных автоматов и докажем для них теорему о развитии маршрутов.

Определим бинарное отношение $\uparrow_A$ на множестве маршрутов конечного автомата A: $(T,T) \in \uparrow_A$ тогда и только тогда, когда $T = T_1T_2T_3$, $T' = T_1T_3$, T_2 есть цикл.

Пусть T, T' — это маршруты конечного автомата A. Пусть либо $T' = T$, либо для некоторого $k > 0$ существует последовательность $gen(T) = (\langle T_{11},T_{12},T_{13}\rangle,\ldots,\langle T_{k1},T_{k2},T_{k3}\rangle)$, такая, что $T' = T_{11}T_{13}$, $T = T_{k1}T_{k2}T_{k3}$, $(T,T_{11}T_{12}T_{13}) \in \uparrow_A$ с историей $\langle T_{11},T_{12},T_{13}\rangle$ и $(T_{(i-1)1}T_{(i-1)2}T_{(i-1)3},T_{i1}T_{i2}T_{i3}) \in \uparrow_A$ с историей $\langle T_{i1},T_{i2},T_{i3}\rangle$, $1 < i \le k$. Тогда назовем последовательность $gen(T)$ *родословной* с предком T' маршрута T; число k назовем *длиной родословной*. Если $T = T'$, то родословная маршрута T (с предком T') пуста и имеет длину 0.

Пусть маршрут T конечного автомата A содержит цикл. Тогда назовем такой маршрут T производным. Маршрут, не являющийся производным, назовем элементарным.

Теорема о развитии маршрутов для конечных автоматов. Пусть A — конечный автомат. Для каждого его маршрута существует такой канон $T_0 \in Core(A, 1)$, что $(T_0,T) \in \uparrow_A^*$. Для каждого элемента $\langle T_1,T_2,T_3\rangle$ родословной маршрута T с предком T_0 машрут T_2 содержится в некотором каноне из $Core(A, 1)$.

Доказательство. Для всех простых маршрутов КА A утверждение теоремы справедливо по определению. Пусть T — производный маршрут A. Тогда $T = T_1T_2T_3$ и T_2 есть цикл (следовательно $(T_1T_3,T) \in \uparrow_A$). Так как $|T_1T_3| < |T|$, существует конечная последовательность $T_0,\ldots,T_k$, такая что $k > 0$, T_0 не содержит циклов, $T_k = T$, $(T_{i-1},T_i) \in \uparrow_A$ для $i = 1,\ldots,k$. Итак, $(T_0,T) \in \uparrow_A^k$, T_0 является каноном, и первая часть теоремы верна. Для каждого элемента $\langle T_1,T_2,T_3\rangle$ родословной маршрута T верно, что $(T_1T_3,T_1T_2T_3) \in \uparrow_A$, а T_2 является циклом. $Core(A, 1)$ — множество 1-канонов конечного автомата A, следовательно, в этом множестве содержатся каноны со всеми участками, которые являются циклами конечного автомата A. Тогда маршрут T_2 содержится в одном из таких канонов.

Таким образом, теперь понятия отношения развития, предка маршрута, родословной, производного маршрута определены для графовых описаний.

3. Сопряжения графовых описаний

Пусть G — графовое описание. *Компонентой сопряжения* ГО G назовем его подграф с выделенными вершинами — начальной V_{beg} и заключительной V_{end} , такой, что выполняется одно из следующих условий:

(1) V_{beg} совпадает с начальной вершиной ГО G; V_{end} совпадает с одной из заключительных вершин ГО G или существует маршрут $T\pi$, такой что $beg(T) = V_{beg}$, $end(T) = V_{end}$, $\omega(\pi)\neq\varepsilon$ (т.е. из V_{end} исходит дуга с непустой пометкой); множества вершин и дуг компоненты определяется множеством маршрутов: $\{T \in Core(G, 2) \mid beg(T) = V_{beg}, end(T) = V_{end}, \omega(T) = \varepsilon\}$;

(2) существует дуга π_1, такая что $beg(\pi_1) = V_{beg}$, $\omega(\pi_1) \neq\varepsilon$; существует маршрут $\pi_1 T\pi_2$, такой что $end(T) = V_{end}$, $\omega(T) = \varepsilon$ и $\omega(\pi_2) \neq\varepsilon$ или маршрут $\pi_1 T$, такой, что $end(T) = V_{end}$, а V_{end} принадлежит множеству заключительных вершин графа G, $\omega(T) = \varepsilon$; вершины и дуги компоненты C определяются множеством маршрутов: $\{\pi_1 T \in Core(G) \mid end(T) = T_{end}, \omega(T) = \varepsilon\}$.

Пометку компоненты сопряжения будем обозначать $\omega(C)$.

Пусть T_1 и T_2 — предложения ГО G_1 и G_2 соответственно, и $\omega(T_1) = \omega(T_2) = x = a_1a_2a_3...a_n$, $n \geq 0$. Тогда *сопряжением* T_1 и T_2 назовем последовательность $S_0,S_1,...,S_n$ пар компонент сопряжения ГО G_1 и G_2 , где $S_i = (C1_i, C2_i)$, для $N = 1, 2$ CN_0 — начальная непомеченная компонента графа G_N (т.е. $\omega(CN_0) = \varepsilon$), для $i = 1,...,n$ CN_i — компонента сопряжения ГО G_N с пометкой a_i (т.е. $\omega(CN_i) = a_i$). Заключительная вершина компоненты CN_i совпадает с начальной у CN_{i+1}, и графовое описание, полученное конкатенацией всех компонент $CN_0...CN_n$, реализует предложение T_N. Пару S_i будем называть *элементом сопряжения*.

Заметим, что в любом сопряжении маршрутов компоненты из элемента S_0 удовлетворяют пункту (1) определения компоненты сопряжения, все остальные компоненты — пункту (2).

Пусть G_1 и G_2 — графовые описания языков, такие что $L(G_1) = L(G_2)$. Тогда *графом сопряжения* назовем граф, дугами которого являются элементы сопряжений предложений графов G_1 и G_2, а вершинами — пары (V_1, V_2), являющиеся начальными или

заключительными вершинами соответствующих компонент некоторого элемента сопряжения.

Алгоритм построения множества компонент сопряжений по конечному множеству предложений.

Действие алгоритма будем обозначать функционально как $S = ConjSet(G_1,Z_1,G_2,Z_2)$.
Вход: эквивалентные графовые описания G_1, G_2, конечное множество Z_1 предложений ГО G_1, конечное множество Z_2 предложений ГО G_2.

Выход: множество S элементов сопряжений.

Метод:

Шаг 1: Положить $S = \varnothing$. Все элементы множеств Z_1 и Z_2 считать необработанными.

Шаг 2: Если в множествах Z_1 и в Z_2 не осталось необработанных элементов, то завершить работу с результатом S.

Шаг 3: Если в Z_1 остался необработанный элемент T_1, то выбрать его для обработки, иначе перейти к шагу 5.

Шаг 4: Обработать T_1 следующим образом. Получить предложение T'_2 ГО G_2, отвечающее цепочке $\omega(T_1)$, $|\omega(T_2)| = n$ и построить сопряжение $S_0,S_1,...S_n$ предложений T_1 и T'_2. Все элементы сопряжений, входящие в это сопряжение, добавить к множеству S. Отметить T_1 как обработанный элемент.

Шаг 5: Если в Z_2 остался необработанный элемент T_2, то выбрать его для обработки, иначе перейти к шагу 7.

Шаг 6: Обработать T_2 следующим образом. Получить предложение T'_1 ГО G_1, отвечающее цепочке $\omega(T_2)$, $|\omega(T_2)| = n$ и построить сопряжение $S_0,S_1,...S_n$ предложений T'_1 и T_2. Все элементы сопряжений, входящие в это сопряжение, добавить к множеству S. Отметить T_2 как обработанный элемент.

Шаг 7: Перейти к шагу 2.

Алгоритм построения элементов сопряжений двух эквивалентных графовых описаний.

Действие алгоритма будем обозначать функционально как $S = Conj(G_1,G_2)$.

Вход: Эквивалентные графовые описания G_1 и G_2

Выход: Множество S элементов сопряжений.

Метод:

Шаг 1: Положить $S_0 = \varnothing$, $i = 0$.
Шаг 2: Увеличить i на 1. Построить множество

$$S_i = ConjSet(G_1, Core(G_1, i), G_2, Core(G_2, i)).$$

Шаг 3: Если $S_i = S_{i-1}$, то завершить работу с результатом $S = S_i$.
Шаг 4: Перейти к шагу 2.

Завершаемость этого алгоритма следует из того, что на каждом его шаге увеличивается подмножество конечного множества всех возможных элементов сопряжений двух заданных графовых описаний.

Заметим, что в случае, если G_1 и G_2 — детерминированные графовые описания и хотя бы одно из них является конечным автоматом, то множество компонент сопряжений $Conj(G_1, G_2)$ является достаточным для построения графа сопряжений G_1 и G_2. В общем же случае, как следствие из основного результата работы [3], можно показать, что для двух детерминированных ГО задача построения графа сопряжений является алгоритмически неразрешимой.

4. Заключение

Итак, графовые описания позволяют единообразно рассматривать описания регулярных и контекстно-свободных языков. На их основе возможно построение более простых и ясных алгоритмов для решения известных задач теории формальных языков для тех подклассов языков, в которых эти задачи алгоритмически разрешимы.

Литература

1. Евстигнеев В.А. Применение теории графов в программировании. М.: Наука, 1985.
2. Станевичене Л.И. К теории бесконтекстных языков. М.: МГУ им. М.В. Ломоносова. 2000. Деп. в ВИНИТИ 29.05.2000. № 1546-В2000.
3. Станевичене Л.И. Одна неразрешимая алгоритмическая проблема // Изв. вузов. Матем. 1996. № 6, стр. 70–77.

Арефьев Н.В., Мальковский М.Г.

Синтаксический анализатор Treevial. Оценка семантической корректности синтаксической структуры

Введение

Алгоритм автоматического синтаксического анализа, лежащий в основе анализатора Treevial, базируется на идее перебора под управлением эвристической функции. Подробно этот алгоритм описан в [1,2]. В данной статье обсуждается проблема оценки правильности синтаксических структур, возникающих в процессе перебора, и предлагается эвристика, позволяющая обнаружить большую часть неправильных структур, а значит сократить время и улучшить качество анализа за счет отсеивания таких структур.

Вначале напомним основные принципы работы синтаксического анализатора Treevial. На вход синтаксического анализатора поступают интерпретации слов, построенные морфологическим анализатором – аннотации (набор пар <атрибут, значение>). Для описания синтаксической структуры предложения в Treevial используется формализм, обобщающий формализм аннотаций – тринотации. Тринотация – это аннотация, которой приписан лес (множество корневых деревьев), в узлах которого стоят другие тринотации, а дуги помечены названиями синтаксических связей. Заметим, что поступающие на вход анализатора аннотации удовлетворяют этому определению и называются тривиальными тринотациями. Процесс синтаксического анализа представляет собой эвристический перебор, на каждом шаге которого к множеству тринотаций применяется одно из заложенных в системе синтаксических правил, связывающих тринотации синтаксическими связями и строящих новые тринотации. Конечная цель перебора – построение упорядоченного набора тринотаций, задающего варианты синтаксического анализа предложения.

Важнейшим фактором, влияющим на скорость и качество работы синтаксического анализатора, является выбор оценочной функции, показывающей, насколько та или иная промежуточная структура близка к правильной синтаксической структуре предложения. Оценка промежуточных структур осуществляется аппаратом штрафных функций (наибольшую оценку получают наименее перспективные структуры). Итоговая оценка синтаксической структуры представляет собой евклидову норму штрафного вектора, компоненты которого отражают степень соответствия синтаксической структуры тем или

иным ограничениям и подсчитываются специализированными штрафными функциями.

Часть штрафных функций можно объединить в класс так называемых «топологических» штрафов, они отражают ограничения на топологию синтаксической структуры (например, штрафуются непроективные структуры, структуры с пересекающимися стрелками синтаксических связей). Эти ограничения описаны в [1,2] и более подробно в [3]. Топологические штрафы не учитывают конкретное лексическое наполнение оцениваемой структуры, что позволяет использовать их в отсутствие словарной информации о сочетаемости слов. Однако, во многих случаях чтобы отличить правильные структуры от неправильных топологических штрафов оказывается недостаточно. Рассмотрим следующие фрагменты предложений: *съесть пирог с черникой, съесть пирог с удовольствием*. Очевидно, что в первом случае предложная группа *с черникой* связана с существительным *пирог* (*пирог с черникой*), во втором же предложная группа *с удовольствием* связана с глаголом *съесть* (*съесть с удовольствием*). Однако анализатор построит по две синтаксические структуры для каждого фрагмента и, если не используется информацию о сочетаемости, не сможет отбросить неправильные синтаксические структуры (*съесть с черникой* и *пирог с удовольствием*).

1. Ограничения на сочетаемость

В лингвистике хорошо известно, что слова в предложении сочетаются не свободно, а в соответствии с множеством различных ограничений [4]. Среди прочих выделяют морфо-синтаксические, лексические и семантические ограничения на сочетаемость. К морфо-синтаксическим ограничениям на сочетаемость некоторого слова относятся, например, ограничения на предложно-падежную форму зависимых слов (*есть кашу*, но *питаться кашей*; *отлынивать от работы*, но *избегать работы)*. Лексические ограничения на сочетаемость выражаются в том, что некоторые слова могут иметь в качестве зависимых только определенные лексемы, причем зачастую выделить общие признаки этих лексем оказывается невозможно (например, в словосочетаниях *уменьшать / сбрасывать скорость, давление, громкость* глаголы *уменьшать / сбрасывать* имеют близкие значения, однако разную лексическую сочетаемость: можно *уменьшить*, но не *сбросить длину, количество, сопротивление*). О семантических ограничениях на сочетаемость слова А говорят в том случае, когда любое слово В, имеющее некоторый семантический признак, способно сочетаться с А. В качестве примера можно привести глаголы *ухудшаться / улучшаться*, которые сочетаются с названиями способностей, процессов, состояний (*слух улучшился, настроение*

ухудшилось), но не конкретных вещей (*утюг стал хуже*, но не *утюг ухудшился*).

Для того, чтобы учитывать подобные ограничения, была введена отдельная компонента штрафного вектора и разработан алгоритм ее вычисления. Этот алгоритм назван алгоритмом оценки семантической корректности синтаксической структуры, он позволяет учесть все перечисленные типы ограничений.

2. Алгоритм оценки семантической корректности синтаксической структуры

На входе алгоритм оценки семантической корректности получает некоторую синтаксическую структуру, на выходе алгоритм генерирует неотрицательное действительное число (семантическую компоненту штрафного вектора входной структуры).

Пусть задано множество *Terms* (назовем его множеством терминов), на нем определена функция $\rho(x,y)$, сопоставляющая любой паре элементов этого множества неотрицательное вещественное число. Для удобства будем называть эту функцию расстоянием между элементами множества. Необходимо, однако, отметить, что функция $\rho(x,y)$ не является расстоянием в строгом математическом смысле; например для нее может не выполняться аксиома тождества ($\rho(x,y)=0$ тогда и только тогда, когда $x=y$).

Пусть также задано множество четверок вида <*templ, Acts, N, C*> (правила приписывания терминов тринотациям), где *Acts: Rels* $\rightarrow$ *Templs* × *Terms* – отображение, ставящее в соответствие каждой синтаксической связи rel_i некоторого подмножества *Rels* множества всех синтаксических связей пару <$templ_i$, C_i>; *templ*, $templ_i$ – множества пар вида <атрибут, значение> (они выступают как шаблоны тринотаций); C_i, $C \in Terms$; *N* – подмножество *Rels*.

Синтаксическая структура представляет собой дерево, в вершинах которого находятся тринотации (как тривиальные, т.е. покрывающие отдельные слова, так и нетривиальные, соответствующие словосочетаниям и имеющие внутреннюю структуру), а дуги помечены названиями синтаксических связей (выделяются также служебные дуги, помеченные символом «*w*», соединяющие нетривиальные тринотации с корневыми узлами вложенных в них структур). Будем приписывать вершинам этого дерева элементы множества <*Terms*>, а дугам – неотрицательные вещественные числа следующим образом:

1. Для каждого листа дерева, соответствующего тринотации с атрибутами *ATTR*, найдем четверку <*templ, Acts, N, C*>, где *templ* $\subseteq$ *ATTR* и $N = \varnothing$; припишем соответствующему листу *C* из этой четверки. Если подходящая четверка не была найдена, припишем листу ε.

2. Рассмотрим некоторую нелистовую вершину, соответствующую тривиальной тринотации с атрибутами *ATTR*; важно, что из таких вершин не могут исходить служебные дуги, помеченные *w*. Пусть каждому ее потомку t_i $(1 \le i \le n)$ с атрибутами $ATTR_i$, соответствующему дуге, помеченной синтаксической связью r_i, уже приписан $C'_i \in Terms$ или ε (в противном случае запускаем процедуру пометки для этого потомка). Найдем четверку $<templ, Acts, N, C>$, где $templ \subseteq ATTR$, $\{r_i\}_1'' \supseteq N$, для каждой связи r_i значение $Acts(r_i)$ либо неопределенно, либо равно $<templ_i, C_i>$ и удовлетворяет условию $templ_i \subseteq ATTR_i$. Если такая четверка найдена, припишем рассматриваемой вершине *C* из этой четверки, иначе – ε. При этом дуге r_i приписывается число, равное $\rho(C'_i, C_i)$, либо 0, если t_i приписан ε или значение $Acts(r_i)$ не определено.
3. Если рассматриваемой вершине соответствует нетривиальная тринотация, из нее выходит по крайней мере одна дуга, помеченная как *w*. Если такая дуга ровно одна, вершина помечается так же, как потомок, соответствующий этой дуге; в противном случае – ε. Дугам, исходящим из вершины, приписываются нули.

Когда рассмотрены все вершины дерева, вычисляется величина ψ, равная сумме чисел, приписанных дугам.

Заметим, что описанная процедура приписывания меток вершинам и дугам дерева является в общем случае недетерминированной: для каждой вершины может быть найдено несколько правил приписывания меток. Рассматриваются все возможные результаты работы процедуры, после чего среди них выбирается результат с минимальной величиной ψ, эта величина и становится оценкой исходной синтаксической структуры.

Поясним суть описанного алгоритма. Основная идея состоит в том, чтобы сопоставить тринотациям их смысловые интерпретации (термины), а затем для каждой синтаксической связи оценить, насколько хорошо ее зависимый член при заданной интерпретации удовлетворяет ограничениям на сочетаемость главного члена. Понятие термина (смысловой интерпретации) в данном случае определяется формально (термин – элемент множества *Terms*), однако оно во многом схоже с тем, что называется *означаемым* у Де Соссюра или *Смыслом* у И.А. Мельчука [5] (понятие термина соотносится с тринотацией как означаемое с означающим или Смысл с Текстом, т.е. как внутреннее содержание языкового знака с внешним проявлением этого знака). Тем не менее, имеет место и важное отличие от упомянутых теорий: термин не обязан описывать значение слова или словосочетания во всех подробностях, во многих случаях для решения поставленной задачи

достаточно указать лишь семантический класс (СОСТОЯНИЕ для слова *настроение*, или СПОСОБНОСТЬ для слова *слух*).

Правила приписывания терминов тринотациям следует интерпретировать следующим образом: некоторой тринотации t может быть приписан термин C, если выполнены следующие условия:

1. t соответствует шаблону *templ* (то есть все атрибуты, присутствующие в *templ*, входят в t с теми же значениями);
2. для каждой синтаксической связи r_i, исходящей из t, для которой определено значение $Acts(r_i) = <templ_i, C_i>$, тринотация на другом конце связи соответствует шаблону $templ_i$;
3. среди синтаксических связей, исходящих из t, присутствуют все связи из N.

В результате обработки некоторой словоформы морфологическим анализатором строится множество тривиальных тринотаций, описывающих варианты морфологического анализа этой словоформы. Среди прочих выделим один атрибут этих тринотаций – идентификатор соответствующей словарной статьи морфологического словаря, с которым работает анализатор. Будем называть эту словарную статью вокабулой, соответствующей тривиальной тринотации (или варианту морфологического анализа). Таким образом, вокабула описывает некоторое множество словоформ и соответствующие им морфологические характеристики: начальную форму, часть речи и другие. Тем не менее, не всегда этому множеству словоформ можно однозначно приписать смысловое толкование. Например, в морфологическом словаре Зализняка, используемом Treevial, слову «*пионер*» соответствует только одна словарная статья, хотя это слово имеет несколько значений (например, ПЕРВОПРОХОДЕЦ и ЧЛЕН_ДЕТСКОЙ_ОРГАНИЗАЦИИ).

Мы ограничимся случаями, когда шаблон тринотации *templ* накладывает ограничения только на значение идентификатора, то есть описывает некоторое множество вокабул, которым при определенных условиях можно сопоставить некоторый термин. Эти условия описываются остальными компонентами правила и представляют собой ограничения на синтаксические связи (тип связи и шаблон дочерней тринотации). Однако таким образом нельзя сопоставить термин нетривиальной тринотации (поскольку такая тринотация не соответствуют никаким вокабулам, так как не представляет собой вариант морфологического анализа). Если внутренняя структура нетривиальной тринотации состоит из единственного дерева тринотаций, термин, приписанный корню этого дерева, передается этой тринотации. В противном случае считается, что тринотации нельзя приписать термин. Например, тринотации *«грибы и ягоды»* не будет сопоставлен термин. Вероятно, в данном случае этой тринотации

разумно сопоставить, например, термин РАСТЕНИЯ, однако общие правила относительно подобных случаев пока не выработаны.

Необходимо отметить, что на этапе сопоставления термина тринотации не учитываются ограничения на термины, приписанные дочерним тринотациям. Однако эти ограничения отражаются на оценке, приписываемой синтаксической связи: величина этой оценки отражает то, насколько «далек» термин, приписанный процедурой дочерней тринотации, от того термина, который указан в правиле. Если значение $Acts(r_i)$ не определено (то есть в правиле отсутствуют ограничения на связи этого типа), либо тринотации на другом конце связи приписана метка ε (означающая, что неизвестны термины, которые можно сопоставить этой тринотации), то связи r_i приписывается нулевая оценка.

Такая процедура оценивания синтаксических связей соответствует следующей идее: если мы не обладаем информацией о некоторой лексической единице, приходится допускать любые варианты ее сочетания с другими лексическими единицами. Если же такая информация у нас есть, мы считаем ее полной и при несогласованности синтаксической структуры с этой информацией, штрафуем оцениваемую структуру.

Еще раз отметим, что приписывание термина тринотации не является детерминированной процедурой. Рассматриваются все возможные варианты приписывания терминов тринотациям, для каждого из этих вариантов вычисляется сумма всех оценок синтаксических связей, минимальная из этих сумм становится оценкой синтаксической структуры. Такой перебор необходим из-за возможности лексической омонимии (когда некоторое слово, входящее в состав анализируемого предложения, может иметь несколько трактовок, причем правильная трактовка выбирается исходя из контекста).

3. Примеры

Рассмотрим предложение *«В огороде рос лук»*. При морфологическом анализе словоформы *«лук»* порождается единственная вокабула, соответствующая обоим значениям этого слова (оружие и растение). При этом разные значения соответствуют разным элементам ЛУК1 и ЛУК2 множества *Terms*, и существует два правила, каждое из которых сопоставляет порожденную вокабулу своему значению. Кроме того, есть правило, утверждающее, что вокабула *«расти»* может иметь в качестве зависимого члена некоторой синтаксической связи термин РАСТЕНИЕ. В таком случае при оценке синтаксической связи между тринотациями *«рос»* и *«лук»* будет рассмотрено два варианта: при сопоставлении тринотации *«лук»* термина ЛУК1 (оружие) синтаксическая связь будет сильно

оштрафована (конечно, расстояние от термина ЛУК1 до термина РАСТЕНИЕ должно быть достаточно велико), во втором же варианте штраф будет нулевой (расстояние от ЛУК2 до РАСТЕНИЕ должно быть нулевым). При оценке синтаксической структуры будет выбрана вторая трактовка слова *«лук»* и указанная синтаксическая связь не будет оштрафована.

Отметим, что побочным эффектом приведенного алгоритма оценки синтаксической структуры может стать разрешение лексической многозначности. Действительно, если минимальная сумма оценок синтаксических связей достигается только на одном варианте приписывания терминов тринотациям, то можно для многозначных слов выбрать именно те значения, которые соответствуют этому варианту. В любом случае, можно отбросить те значения, которые соответствуют только сильно оштрафованным вариантам приписывания терминов.

В качестве другого примера рассмотрим два предложения: *«Он уехал в деревню на холме»* и *«Он уехал в деревню на электричке»*. В правильных синтаксических структурах этих предложений словосочетание *«на холме»* подчинено существительному *«деревня»* (*«деревня на холме»*), а словосочетание *«на электричке»* – глаголу *«уехать»* (*«уехать на электричке»*). Чтобы отличить эти структуры от неправильных (*«деревня на электричке»* и *«уехать на холме»*) в правилах должно быть указано, что глагол *«уехать»* может быть связан со словосочетаниями, которым приписаны термины НА_ЧЕМ и КУДА, а существительное *«деревня»* – со словосочетаниями, имеющими значение ГДЕ. Предлогу *на* можно сопоставить термин НА_ЧЕМ, если он связан с существительным в предложном падеже со значением СРЕДСТВО_ПЕРЕДВИЖЕНИЯ, либо ГДЕ, если существительное имеет значение МЕСТО. В наименее штрафованных вариантах алгоритм сопоставит словосочетанию *«на электричке»* термин НА_ЧЕМ, а словосочетанию *«на холме»* - термин МЕСТО. Таким образом, будут выбраны правильные синтаксические структуры.

Заключение

В статье предложена эвристика, позволяющая в процессе работы синтаксического анализатора Treevial оценивать генерируемые им синтаксические структуры с точки зрения сочетаемости слов, между которыми проведены синтаксические связи. Эта эвристика во многих случаях позволяет отсеять неправильные синтаксические структуры, что существенно увеличивает скорость работы анализатора и позволяет корректно отсортировать результаты его работы.

В данной статье не указано, откуда берется и каким образом представлена в системе информация о сочетаемости. Этому посвящены текущие исследования, некоторые результаты которых приведены в [6].

Литература

1. Мальковский М.Г., Старостин А.С. Синтаксический анализатор Treevial. Постановка задачи синтаксического анализа. // Программные системы и инструменты. Тематический сборник N9, М.: Изд-во факультета ВМиК МГУ, 2008. – С. 95-108.
2. Старостин А.С. Синтаксический анализатор Treevial. Алгоритм восходящего синтаксического анализа с памятью, работающий под управлением эвристической функции. // Программные системы и инструменты. Тематический сборник N9, М.: Изд-во факультета ВМиК МГУ, 2008. – С. 109-117.
3. Гладкий А.В. Синтаксические структуры естественного языка в автоматизированных системах общения. М.: Наука, 1985. – 144 с.
4. Апресян Ю.Д. Избранные труды, т.1. Лексическая семантика. М.: Школа «Языки русской культуры», Издательская фирма «Восточная литература» РАН, 1995. – 472 с.
5. Мельчук И.А. Опыт теории лингвистических моделей «СМЫСЛ ↔ ТЕКСТ». М.: Наука, 1974. – 316 с.
6. Мальковский М.Г., Арефьев Н.В. Использование онтологии в системе автоматического синтаксического анализа // Сб. научных трудов по материалам международной научно-практической конференции «Современные проблемы и пути их решения в науке, транспорте, производстве и образовании '2008», т.2. Одесса: Черноморье, 2008. – С. 65-67.

Павлов А.С.

Исследование устойчивости метода обнаружения поискового спама на основе статистических характеристик

1. Введение

В настоящее время в сети Интернет поиск является основным источником информации. Отвечая на запрос пользователя, поисковая машина выдает только ограниченное число результатов на странице выдачи. Из-за этого возникает конкуренция за попадание на первую страницу выдачи поисковых систем. Зачастую недобросовестные пользователи пытаются повлиять на алгоритмы, использующиеся для выбора документов, наиболее релевантных запросу. Для этого они массового создают сайты, единственная цель которых, повлиять на выдачу поисковой системы. Это явление называется поисковым спамом и представляет одну из наибольших угроз для поисковых машин.

По оценкам поисковый спам составляет до 22% [8] содержимого всей сети Интернет. При этом большинство сайтов, являющихся поисковым спамом, не представляют ценности для пользователей. Поисковый спам ухудшает качество поиска, а также приводит к заполнению индексов поисковых систем бесполезными документами, которые не следовало бы индексировать.

Поисковый спам эффективен при массовом создании спам-страниц. При этом наличие на таких страницах уникального содержимого затрудняет их обнаружение. Одним из наиболее распространенных методов порождения уникальных текстов являются генераторы текстов на основе цепей Маркова. При порождении текстов с помощью таких генераторов сначала на отобранных текстах-образцах производится обучение, затем по образцам строится большое количество бессмысленных, но локально связных текстов. Помимо локальной связности такие тексты также приблизительно соответствуют тематике исходных документов-образцов.

В рамках данной работы излагается подход к обнаружению текстов, обладающих локальной связностью, но нарушающих другие закономерности, характерные для естественных текстов [1]. Предметом исследования в данной работе является значимость различных статистических характеристик текстов, применяемых при обнаружении поискового спама. Также изучается устойчивость предлагаемого метода к попыткам его скомпрометировать, имитацией некоторых характеристик реальных текстов.

2. Обзор существующих методов

2.1. Методы детектирования поискового спама

Применимость простых статистических характеристик для определения поискового спама изучалась в работе [11]. При этом наибольшее внимание уделялось ссылочным характеристикам, в то время как статистические характеристики текста практически не рассматривались. Исследование лингвистических характеристик для обнаружения поискового спама исследовалось в работе [14]. Данный подход основывается на применении специального словаря, на основе которого вычисляется более 200 статистических признаков. Применение словаря означает, что некоторые генераторы текстов могут обойти предлагаемые методы.

Еще один интересный подход к анализу содержимого документов для обнаружения поискового спама предлагается в [6]. Эта работа основывается на определении коммерческой направленности текстов по нескольким статистическим атрибутам.

Подходы, не зависящие от конкретной лексики и тематики документов, предлагаются в работах [13, 16]. Первая работа посвящена обнаружению спама в блогах, и использует особенности формата блогов, например, наличие комментариев, что ограничивает применимость данного метода. Вторая работа основана на анализе стилистических особенностей HTML-кода страниц, в то время как текстовое содержимое не учитывается в принципе.

2.2. Использование несловарных характеристик при анализе текстов

Существует постоянный интерес исследователей к анализу статистических, трудно контролируемым автором, характеристик естественного текста.

Метрики читаемости (readability, также употребляется термин «читабельность») текста подробно описаны в [9, 10]. Применение простых статистических характеристик, таких как длина предложений и длина слов, широко используется в США для оценки простоты восприятия текста.

Задача определения жанра текста по простым статистическим характеристикам решается в [7]. В этой работе показано как небольшое число характеристик текста позволяют с неплохой точностью определять наиболее распространенные стили и жанры.

Определение авторства текста, точнее специфического авторского стиля, также может основываться на глобальных статистических закономерностях текстов. Большое количество работ в данном направлении основывается на статье [5]. Анализ статистики употребления частиц, предлогов, а также длин предложений и слов

позволяет формулировать критерии принадлежности текста конкретному автору.

2.3. Определение дубликатов

Задача обнаружения дубликатов текста является смежной с задачей обнаружения текстов, созданных цепями Маркова, так как в зависимости от длины цепи порожденный текст может копировать большие куски документов-образцов.

Обзор методов обнаружения дубликатов приведен в [2]. В статье [12] описано применение шинглирования для обнаружения спам-текстов, порожденных из отрывков естественных текстов.

3. Метод обнаружения поискового спама на основе статистических характеристик текста

В основе рассматриваемого подхода лежат методы, также использовавшиеся для определения жанра текста и авторства. В данном методе выделяется набор трудно контролируемых автором статистических признаков текстового документа. Затем, на основе полученных характеристик и машинного обучения строится автоматический классификатор, который позволяет обнаруживать неестественные тексты.

Данный метод основывается на предположении, что по выбранным характеристикам тексты, полученные с помощью генератора на основе цепей Маркова, будут отличаться от естественных текстов. Применяемые характеристики можно разделить на три группы:

- Глобальные статистические характеристики текста;
- Статистика употребления частей речи;
- Характеристики разнообразия текста;
- Статистика употребления редких оборотов.

3.1. Глобальные статистические характеристики текстов

Глобальные статистические характеристики зачастую используются при оценке читаемости текстов. Признаки, попавшие в данную группу, были выбраны, потому что их трудно контролировать человеку-автору. К тому же, ряд нижеперечисленных характеристик сильно коррелируют с читаемостью текста [10]. Неестественные тексты практически всегда нечитаемые и лишены смысла, поэтому использование этих характеристик может быть полезно при их обнаружении.

В данную группу попадают следующие параметры:

- Средняя длина слов в буквах;
- Средняя длина слов в слогах;

- Доля слов длиннее 7 букв;
- Доля слов длиннее 7 слогов;
- Доля слов менее чем из 3 слогов;
- Доля односложных слов;
- Среднее количество знаков пунктуации на предложение;
- Средняя длина предложений в словах;
- Минимальное количество слов в предложении;
- Максимальное количество слов в предложении;
- Среднее количество слов, начинающихся с большой буквы в предложениях.

3.2. Статистика употребления частей речи

Каждый человек обладает уникальным авторским стилем. Каждому автору свойственно употреблять некоторые части речи чаще, чем другие. Признаки, связанные со статистикой употребления частей речи, используются при определении авторства текстов [5]. Эти же характеристики можно использовать для обнаружения результатов работы генераторов текста.

К данной группе относятся следующие характеристики:

- Среднее количество глаголов в предложениях;
- Среднее количество существительных в предложениях;
- Среднее количество прилагательных в предложениях;
- Среднее количество существительных в родительном падеже в предложениях;
- Среднее количество существительных мужского рода в предложениях;
- Среднее количество существительных женского рода в предложениях;
- Среднее количество существительных среднего рода в предложениях;
- Доля существительных мужского рода;
- Доля существительных женского рода;
- Доля существительных среднего рода;
- Доля глаголов прошедшего времени;
- Доля глаголов непрошедшего времени;
- Доля глаголов настоящего времени;
- Доля предлогов в тексте;
- Дисперсия доли предлогов в предложениях;
- Дисперсия доли существительных в предложениях;
- Дисперсия доли глаголов в предложениях;
- Доля предложений с несколькими глаголами.

3.3. Характеристики разнообразия текста

Одной из характерных особенностей естественных текстов является ограниченность словаря наиболее используемых слов. При этом частоты употребления слов подчиняются закону Ципфа, по которому, если упорядочить слова текста по частотности, то частота каждого слова будет обратно пропорциональна его порядковому номеру.

Частота f слова с порядковым номером k подчиняется следующему соотношению:

$$f(k;s,c) \approx \frac{c}{k^s};$$

где s – параметр, характеризующий разнообразие слов в тексте, c – параметр, характеризующий частоту наиболее популярных слов. Для оценки разнообразия слов в тексте можно по частотам слов в тексте оценить параметры s и c.

Другим способом оценить разнообразие текста является измерение степени сжатия текста различными алгоритмами сжатия. Чем более разнообразным является словарный запас автора текста, тем слабее тексты сживаются алгоритмами сжатия.

Еще одной характеристикой, относящейся к данной группе, является доля слов, повторяющихся в соседних предложениях. В естественных текстах доля таких слов высока, так как для поддержания логической связности текста автор вынужден повторять некоторые слова в соседних предложениях. В итоге, в данной группе были выбраны следующие характеристики:

- Параметр s в распределении Ципфа для слов;
- Параметр c в распределении Ципфа для слов;
- Параметр s в распределении Ципфа для существительных;
- Параметр c в распределении Ципфа для существительных;
- Среднее количество слов, повторяющихся в соседних предложениях;
- Степень сжатия текста алгоритмом bz2;
- Степень сжатия текста алгоритмом gzip.

3.4. Статистика употребления редких оборотов

Некоторым жанрам естественных текстов свойственно ограниченное употребление некоторых частей речи или оборотов [7]. Например, в нормативно-правовых актах крайне редко встречаются частицы «ну» и «вот», так как использование этих частиц противоречит стилю, принятому в таких документах. В связи с этим редко употребляемые слова и обороты могут служить хорошим индикатором стиля текста.

Так как генераторы текстов напрямую не моделируют стилистическое единство порождаемого текста, редкие характеристики могут использоваться для обнаружения искусственных текстов. К данной группе характеристик относятся:

- Доля частиц в тексте;
- Доля междометий в тексте;
- Доля местоименных существительных в тексте;
- Доля местоименных прилагательных в тексте;
- Доля местоименных наречий в тексте;
- Доля числительных в тексте;
- Доля местоимений первого лица в тексте;
- Доля местоимений второго лица в тексте;
- Доля глаголов в повелительном наклонении;
- Доля деепричастий;
- Доля частиц «бы»;
- Доля частиц «ну», «вот», «ведь»;
- Среднее количество вводных слов в предложениях;
- Дисперсия доли частиц в предложениях;
- Дисперсия доли междометий в предложениях;
- Дисперсия доли местоименных прилагательных в предложениях;
- Дисперсия доли местоименных существительных в предложениях;
- Дисперсия доли числительных в предложениях;
- Дисперсия доли местоименных наречий в предложениях;
- Дисперсия доли порядковых числительных в предложениях;
- Среднее количество знаков экспрессивной пунктуации («!», «?», «...») в предложениях.

3.5. Метод машинного обучения

Для каждого документа извлекаются все перечисленные характеристики. Каждому документу ставится в соответствие вектор признаков, в котором сохраняются все выделенные характеристики. Затем автоматический классификатор использует построенные вектора признаков для определения документов, порожденных с помощью генераторов текстов.

Для построения автоматического классификатора был предложен алгоритм построения деревьев решений, на основе алгоритма C4.5 [15]. Подробное описание применявшегося метода машинного обучения приведена в [1].

3.6. Программная реализация

Предложенный подход был реализован в виде двух программных модулей:

- Модуль обработки текстов;

- Модуль машинного обучения.

Модуль обработки текстов принимает на вход HTML документ. Для данного документа он производит выделение текста. Выделенный текст обрабатывается с помощью парсера mystem [4], в котором происходит определение частей речи, определение границ предложений, а также приведение слов к базовым словоформам. По обработанному тексту собираются вышеописанные характеристики, полученный вектор признаков сохраняется.

Модуль машинного обучения работает в двух режимах. В режиме обучения он принимает на вход тренировочный набор векторов признаков, а также метки, соответствующие этому набору признаков. При этом строятся деревья решений и формируется классификатор. В режиме классификации ранее построенный классификатор принимает на вход вектор признаков и выдает вес каждой из меток для данного вектора признаков.

4. Эксперименты

Предложенный подход и набор характеристик были апробированы в ходе нескольких экспериментов.

4.1. Тренировочный набор

Эксперимент проводился на коллекции веб страниц ROMIP By.Web [3]. В качестве модельного генератора текстов был реализован генератор текстов на основе цепей Маркова. Тренировочные наборы для классификаторов строились следующим образом:

1. Из набора By.Web были взяты 20000 произвольных документов, помеченных как good (GoodSet);
2. Из набора By.Web были взяты другие 20000 произвольных документов (ExampleSet).
3. Используя набор документов ExampleSet в качестве образца для алгоритма генерации текстов, был порожден набор из 20000 искусственных текстов, помеченных как spam (SpamSet).

Тренировочный набор был создан из 10000 документов из GoodSet и 10000 документов из SpamSet. Оставшиеся документы составили тестовый набор. В ходе каждого из экспериментов формировались классификаторы, обученные на тренировочном наборе.

4.2. Эксперимент по обнаружению искусственных текстов

В первом эксперименте оценивалась возможность обнаружения текстов, порожденных цепями Маркова. Для этого классификатор был обучен на тренировочном наборе. Применение полученного классификатора к тестовому набору дало точность определения искусственных текстов 95,03%, полноту 95,77%, и результирующую F-

меру 95,4%. Данный эксперимент подтвердил, что предложенный набор характеристик может быть использован для обнаружения текстов, порожденных цепями Маркова.

4.3. Эксперимент по оценке качества отдельных характеристик

В ходе второго эксперимента оценивался вклад каждого из выделяемых признаков в классификацию. Для этого были построены классификаторы, обученные на тренировочном наборе, из которого удалены все, кроме одного, признаки. В рамках данного эксперимента для каждой характеристики измерялась F-мера обнаружения искусственных текстов при классификации только по данной характеристике. В результате каждой характеристике ставится в соответствие одно число, характеризующее вклад характеристики в машинное обучение.

Результаты эксперимента для десяти наиболее значимых характеристик приведены в таблице 1.

Характеристика текста	**F-мера**
Степень сжатия текста алгоритмом bz2	78,87%
Степень сжатия текста алгоритмом gzip	77,92%
Параметр *s* в распределении Ципфа для существительных	77,67%
Дисперсия доли местоименных наречий в предложениях	75,64%
Дисперсия доли междометий в предложениях	75,23%
Дисперсия доли частиц в предложениях	75,22%
Дисперсия доли предлогов в предложениях	75,14%
Доля местоименных наречий в тексте	74,94%
Дисперсия доли местоименных прилагательных	74,70%
Дисперсия доли местоименных существительных	74,43%

Таблица 1. F-мера классификации при использовании только одной характеристики документа

4.4. Оценка устойчивости алгоритма

Важной свойством методов борьбы с поисковым спамом является их устойчивость к намеренному воздействию на них. В рамках предлагаемого подхода такое воздействие может выражаться в использовании генераторов текста, которые контролируют характеристики порождаемого текста, чтобы затруднить их обнаружение.

В третьем эксперименте исследовалось качество обнаружения порожденных текстов в случае, если спамерам удается порождать тексты, повторяющие одну или несколько характеристик естественных текстов. В рамках данного предположения считается, что спамеры

пытаются в первую очередь смоделировать те характеристики, которые имеют наибольший вклад в классификацию.

В ходе данного эксперимента из классификации последовательно удалялись наиболее ценные признаки. Для каждого сокращенного набора признаков вычислялась точность, полнота и F-мера обнаружения текстов, порожденных генератором. Графики этих величии в зависимости от количества удаленных из классификации характеристик представлены на рисунке 1.

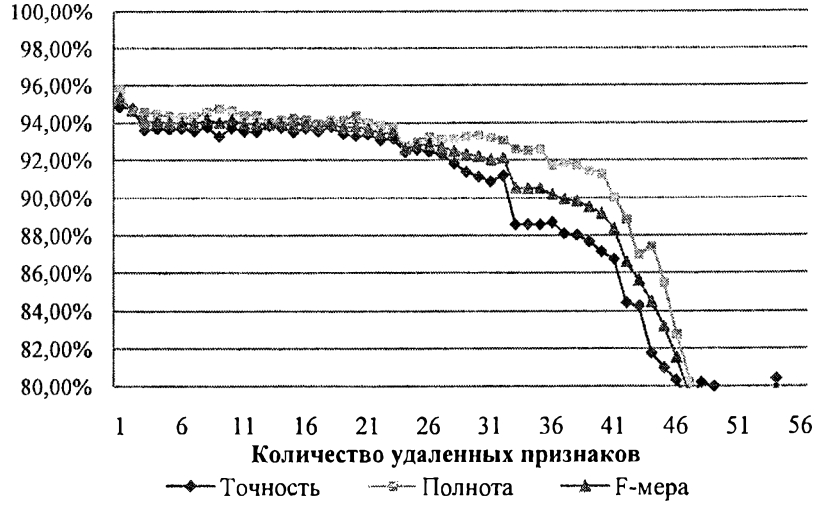


Рисунок 1. Зависимость качества классификации от количество удаленных признаков.

4.5. Поиск минимального набора характеристик

Еще одной важной задачей является поиск минимального набора признаков, при котором сохраняется приемлемое качество классификации. Выделение минимального набора позволяет уменьшить затраты на построение векторов признаков, а также уменьшает объем данных, которые надо хранить для каждого документа. Эти характеристики особенно важны в промышленных поисковых системах, где индексируются миллиарды документов.

Эксперимент по поиску минимального набора проводился следующим образом: из всего набора признаков последовательно удалялись характеристики, начиная с характеристик наименее ценных для классификации. Когда, после удаления очередного признака, F-мера классификации опускалась ниже определенного порога, данный признак обозначался как принадлежащий минимальному набору, и

снова добавлялся в общий набор признаков. В рамках данного эксперимента минимальный порог для F-меры был выбран равным 94%.

В результате был получен следующий минимальный список характеристик:

- Максимальное количество слов в предложении;
- Среднее количество слов, начинающихся с большой буквы в предложениях;
- Среднее количество существительных в предложениях;
- Доля предложений с несколькими глаголами;
- Доля предлогов в тексте;
- Дисперсия доли предлогов в предложениях;
- Параметр *s* в распределении Ципфа для существительных;
- Степень сжатия текста алгоритмом bz2;
- Доля местоименных существительных в тексте;
- Доля местоименных наречий в тексте;
- Дисперсия доли частиц в предложениях;
- Дисперсия доли местоименных прилагательных в предложениях;
- Дисперсия доли местоименных существительных в предложениях;
- Среднее количество знаков экспрессивной пунктуации в предложениях.

Аналогичный подход может быть использован для поиска минимальных наборов признаков для других порогов.

5. Выводы

Результаты экспериментов подтвердили, что предложенный подход может быть использован для обнаружения поискового спама, порожденного с помощью генераторов текстов, сохраняющих лишь локальную связность.

Оценка вклада отдельных признаков показывает, что порожденные тексты сильно отличаются от естественных по разнообразию словарного запаса. Об этом свидетельствует высокое качество классификации при использовании характеристик, оценивающих разнообразие текста. При этом, очевидно, что этих признаков не достаточно для точной классификации.

В рамках данной работы также показана устойчивость метода к попыткам смоделировать характеристики естественных текстов. Даже при удалении половины признаков из классификации F-мера оказывается больше 90%. Высокая устойчивость алгоритма при удалении признаков увеличивает затраты спамеров на создание генераторов текста, которые бы обходили данный метод, так как им потребуется учитывать большое количество ограничений при порождении текстов. Открытым остается вопрос о принципиальной возможности создания такого генератора.

Анализ минимального набора признаков для классификации показывает, что все четыре предложенные группы признаков играют роль при обнаружении искусственных текстов.

Заключение

В рамках данной работы рассмотрен метод обнаружения поискового спама, порожденного с помощью генераторов текста. Данный метод основывается на выделении в текстах большого количества разнородных статистических характеристик, и применении методов машинного обучения для автоматической классификации документов.

Было произведено исследование характеристик, применявшихся в данном методе, которое показало, какие свойства естественных текстов нарушаются в текстах, порожденных генераторами. Также была экспериментально подтверждена устойчивость предложенного метода, в условиях, когда спамеры пытаются сымитировать некоторые из рассмотренных характеристик естественных текстов.

В дальнейшем планируется развитие данного подхода. В данный момент не учитывается, что поисковый спам зачастую располагается на отдельных сайтах, и все документы с таких сайтов являются спамом. С помощью предложенного метода можно с высокой точностью обнаруживать сайты, на которых большинство документов порождены с помощью генераторов текста.

Литература

1. Павлов А.С., Добров Б.В. Методы обнаружения поискового спама, порожденного с помощью цепей Маркова // Труды 11й Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2009, Петрозаводск, Россия, 2009.
2. Зеленков Ю.Г., Сегалович И.В., Сравнительный анализ методов определения нечетких дубликатов для Web-документов // Труды 9ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2007, Переславль, Россия, 2007.
3. Веб коллекция BY.Web, http://romip.ru/ru/collections/by.web-2007.html.
4. Парсер mystem http://company.yandex.ru/technology/mystem/.
5. Фоменко В.П., Фоменко Т.Г., Авторский инвариант русских литературных текстов, 1981.
6. Benczúr, A. A., Bíró, I., Csalogány, K. and Sárlós, T. Web Spam Detection via Commercial Intent Analysis. In Proceedings of the 3rd

international workshop on Adversarial Information Retrieval on the Web, Banff, Alberta, Canada, May 8th, 2007. Pages: 89–92.
7. Braslavski P. Document Style Recognition Using Shallow Statistical Analysis. In Proceedings of the ESSLLI 2004 Workshop on Combining Shallow and Deep Processing for NLP, Nancy, France, 2004, p. 1–9.
8. Castillo, C., Donato, D., Becchetti, L., Boldi, P., Leonardi, S., Santini, M., Vigna, S. A Reference Collection for Web Spam. ACM SIGIR Forum Volume 40, Issue 2 (December 2006) Pages: 11–24.
9. Dale, E. and J. S. Chall. 1949. —The concept of readability.□ Elementary English 26: 23.
10. Dubay, W.H.. 2004. The Principles of Readability. Costa Mesa, CA: Impact Information
11. Fetterly, D., Manasse, M., Najork, M. Spam, damn spam, and statistics: using statistical analysis to locate spam web pages. In Proceedings of WebDB'04, New York, USA, 2004.
12. Fetterly, D., Manasse, M., Najork, M. Detecting phrase-level duplication on the World Wide Web. In Proceedings of SIGIR'05, pages 170–177, New York, NY, USA, 2005. ACM.
13. Mishne, G., Carmel, D., and Lempel, R. Blocking blog spam with language model disagreement. In Proceedings of AIRWeb 2005, May 2005.
14. Piskorski, J., Sydow, M., Weiss, D., Exploring Linguistic Features for Web Spam Detection: A Preliminary Study. In Proceedings of the 4th international workshop on Adversarial Information Retrieval on the Web, Beijing, China, Pages 25-28.
15. Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.
16. Urvoy T., Chauveau E., Filoche, P. Tracking Web Spam with HTML Style Similarities. ACM Transactions on the Web, Vol. 2, No. 1, Article 3.

Раздел VI
Программные системы

Жарков А.В., Пивоварчук Д.Г.

Разработка и исследование параллельного алгоритма решения задачи оптимизации управления развитием инфраструктуры типа поставщик-потребитель

Фак-т. ВМиК МГУ имени М.В.Ломоносова, Москва,
Email: zhalex@yandex.ru, dpivovartchuk@gmail.com

1. Введение

Доклад посвящён исследованию параллельного алгоритма решения оптимизационной задачи на параллельных системах. Под инфраструктурой "поставщик-потребитель" в данной работе понимается сеть поставок, объединяющая некоторое количество поставщиков, потребителей и дистрибьюторов. Поставщики генерируют некоторый поток, под которым, в каждом конкретном случае, можно понимать, например, энергию, энергоносители, товары и т.п. Потоки от поставщиков либо доходит непосредственно до потребителей, либо достигает распределителей, которые перераспределяют входящие в них потоки по нескольким направлениям, после чего потоки доходят до потребителей. Поток, дошедший до потребителя, поглощается им. Инфраструктура представляет собой набор заданных связей между поставщиками, дистрибьюторами и потребителями. Подробнее про инфраструктуры для передачи потоков можно прочитать в [1]. Развитием инфраструктуры будем называть увеличение пропускных способностей связей в течение заданных моментов времени. Под оптимизацией развития сети поставок понимается её развитие с целью максимального удовлетворения спроса потребителей.

2. Формальная модель

Сеть поставок представляет собой направленный граф. Каждая вершина графа является одним из следующих объектов: *поставщик, потребитель* или *дистрибьютор.* Будем обозначать: поставщиков — S_1, ..., SN_s, потребителей — D_1, ..., DN_d и дистрибьюторов — B_1, ..., B_{Nb}.

Направленную связь, соединяющую вершины i, j, и направленную от вершины i к вершине j будем обозначать $C_{i,j}$. Соответствующую этой связи пропускную способность будем

обозначать $c_{i,}$ j, а поток через эту связь будем обозначать $f_{i,}$ j. Например, связь, соединяющую поставщика Si и потребителя Dj обозначим $C_{si,}$ d, ее пропускную способность $c_{si,\ dj}$, а поток $f_{si,\ dj}$.

Будем считать, что выполняются следующие естественные предположения:

a) Если вершина соответствует поставщику, то эта вершина может иметь только исходящие связи с потребителями или дистрибьюторами;

b) Если вершина соответствует потребителю, то эта вершина может иметь только входящие связи от поставщиков или дистрибьюторов;

c) Если вершина соответствует дистрибьютору, то эта вершина имеет по крайней мере одну входящую и одну исходящую связь. Дистрибьютор может иметь входящие связи от поставщиков или других дистрибьюторов и исходящие связи с потребителями или другими дистрибьюторами.

d) Каждый поставщик $S_{i,}$ i =, . . . , $N_{s,}$ генерирует исходящий поток и характеризуется числом $s_{i,}$ которое определяет объем генерируемого им потока в единицу времени. Возможно E различных значений.

e) Каждый потребитель $D_{i,}$ i = 1,..., $N_{d,}$ принимает входящий поток и характеризуется числом $d_{i,}$ которое определяет объем потока, который этот потребитель может принять в единицу времени. Возможно E различных значений.

i) Каждый дистрибьютор $B_{i,}$ i = 1,..., N_{b}, принимаемый входящий поток и превращает его в исходящий поток. Дистрибьютор характеризуется числом $b_{i,}$ которое определяет объем потока, который этот дистрибьютор может перераспределить. Возможно E различных значений.

g) Потоки, произведенные всеми поставщиками, распределяются по сети в соответствии с некоторым правилом, которое будем назвать *статическим правилом распределения потоков.*

h) Суммарный поток, ушедший от поставщиков, в полном объеме доходит до потребителей. Причем, не предполагается, что от потребителей уходит весь поток, который он может сгенерировать.

На каждом от 1 до Q интервале времени мы имеем возможность увеличить пропускные способности связей, причем предполагаем, что связи, чьи пропускные способности могут быть увеличены разбиты на группы (1..Q). К одной группе относятся связи, чьи пропускные способности должны быть увеличены за один и тот же интервал времени.

Для решения задачи оптимизации необходимо решить подзадачи линейного программирования для всевозможных наборов групп и значений параметров s_{i} , $d_{i,}$ b_{i}

$$\sum_{i=1}^{N_d}\left[d_i - \sum_{j\in C_{*,\,d_i}} f_{j,\,d_i}\right] \to \min_{(f_{i,j})}$$

При ограничениях

$$\begin{cases} 0 \le f_{i,\,j} & \le & c_{i,\,j}(\bar{G}), \\ \sum\limits_{j\in C_{s_i,*}} f_{s_i,\,j} & \le & s_i, \\ \sum\limits_{j\in C_{*,\,d_i}} f_{j,\,d_i} & \le & d_i, \\ \sum\limits_{j\in C_{*,\,b_i}} f_{j,\,b_i} & \le & b_i, \\ \sum\limits_{j\in C_{*,\,b_i}} f_{j,\,b_i} & = & \sum\limits_{j\in C_{b_i,*}} f_{b_i,\,j}. \end{cases}$$

после чего решение исходной задачи найти по формуле методом динамического программирования [2],[3]:

$$V_\tau(\bar{G}_\tau,(s_\tau,d_\tau,b_\tau)) = \beta_\tau W(\bar{G}_\tau,(s_\tau,d_\tau,b_\tau)) +$$

$$+\min_{q_\tau\in\bar{G}_\tau}\left\{\sum_{(s_{\tau+1},d_{\tau+1},b_{\tau+1})\in\mathcal{H}} p_{\tau+1}(s_{\tau+1},d_{\tau+1},b_{\tau+1}|s_\tau,d_\tau,b_\tau)V_{\tau+1}(\bar{G}_\tau\cup q_\tau,(s_{\tau+1},d_{\tau+1},b_{\tau+1}))\right\}.$$

Алгоритм решения данной оптимизационной задачи требует большого объёма вычисления и большого объёма данных. Так для Q=32 необходимо 2^{32} *sizeof(float) байт > 2 Гбайт, и, соответственно, нужно решить 2^{32} задач линейного программирования.

В связи с этим, актуальной является решение задачи на высокопроизводительных суперкомпьютерных системах.

Основными параметрами модели будем считать:

$N_1 = N^* (N+N_s+N_d+2^*N_b)$ – число неравенств в одной задаче ЛП

$N_2 = E^*2^Q$ - число задач ЛП

3. Параллельная реализация

Параллельная реализация выполнена на основе интерфейса MPI по принципу «один сервер-процесс – множество клиент-процессов». Для решения задачи линейного программирования была выбрана библиотека программ, использующих широко известные алгоритмы исследования операций, GNU GLPK [4]. Необходимою оптимальную последовательность (q□,..., qN□-i) включения групп связей, соответствующую заданной наблюдаемой последовательности {(st,dt,bt)}t=o,...,N можно визуализировать различными способами, наиболее удобным была избрана последовательность графов. Для построения графов используется библиотека Graphviz [5] — пакет утилит по автоматической визуализации графов, заданных в виде текстового описания.

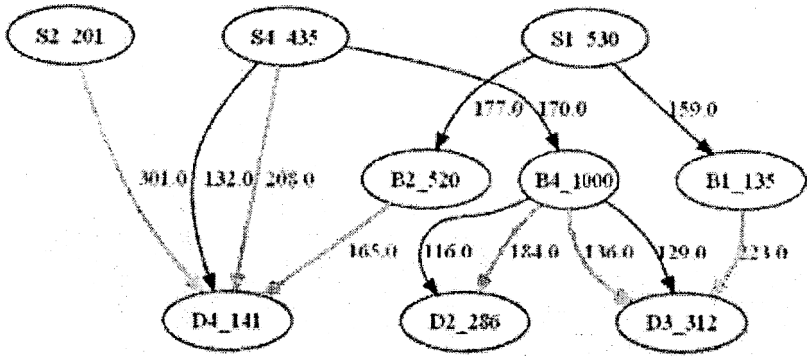


Рис. 1. Визуализация графа с использованием GraphViz

4. Вычислительный эксперимент. Выводы.

Для проведения вычислительного эксперимента на BlueGene/P была написана вспомогательная программа, генерирующая входные модели с заданными параметрами вычислительной сложности N_1 и N_2.

По результатам запуска реализации алгоритма на тестовых моделях были получены следующие графики эффективности и масштабируемости параллельных вычислений, показанные на рис.1 и рис.2, в соответствии с [6],[7] . Были протестированы модели с параметрами N_1 от 400000 (N=40,Ns=10,Nd=10,Nb=10) до 34080000 (N=4045, Ns=1000, Nd=1000,Nb=1000) и N_2 от 2 (E=1, Q=1) до 67108864 (E=1, Q=26). Запуск проводился на числе процессоров от 2 до 2048 IBM BlueGene/P.

В результате эксперимента получены следующие данные:

1. программная реализация позволяет решать задачи с параметрами $1<N_1<32*2^{20}$, $1<N_2<2^{26}$

2. время решения задачи с параметрами $N_1=30*20^{20}$, $N_2=32768$ составило 1674 секунд на 512 процессорах Bluegene/P

3. максимальное достигнутое ускорение на Bluegene/P на задаче $N_1=13107200$, $N_2=8192$ составило 1444 при запуске на 2048 процессорах, эффективность использования процессоров на этой задаче 70% (рисунок 2)

4. прогнозируемое время решения задачи с параметрами $N_1=30*20^{20}$, $N_2=2^{26}$ на 1 процессоре составит 36 лет, применение 2048 процессоров позволит сократить время решения до 9 дней

5. Время решения задачи незначительно изменяется при увеличении её вычислительной сложности в k раз с одновременным увеличением числа процессоров в k раз (рисунок 3), т.е. задача хорошо масштабируется.

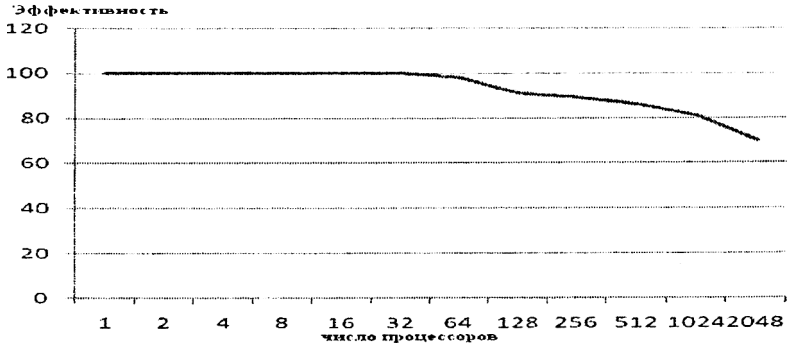


Рис. 2. Эффективность. Падение эффективности с увеличением числа процессоров связано с наличием доли последовательных вычислений

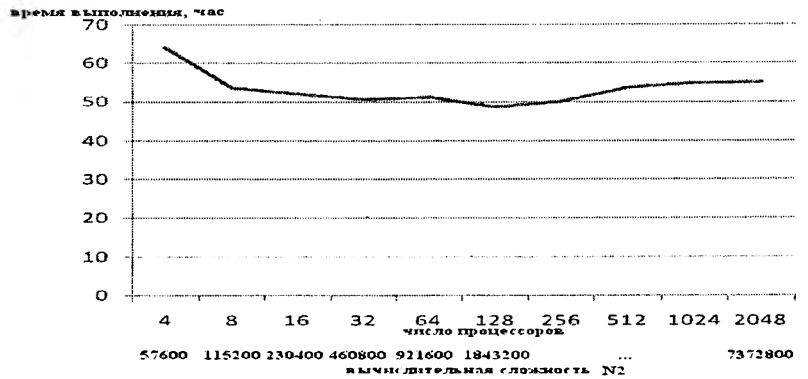


Рис. 3. Масштабируемость. Число процессоров увеличивается пропорционально размерности задачи. Время выполнения показано в часах.

Литература

1. Flow Network [HTML] (http://en.wikipedia.org/wiki/Flow_network)
2. Вентцель Е.С. Элементы динамического программирования. М.: Наука, 1964, 552 с.
3. Беллман Р.Е., Дрейфус С.Е. Прикладные задачи динамического программирования. М.: Наука, 1965, 457 с.
4. Набор средств для линейного программирования GNU (GLPK) [HTML] (http://www.ibm.com/developerworks/ru/opensource/)
5. Graphviz [HTML] (http://lib.custis.ru/index.php/Graphviz)
6. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. СПб.: БХВ-Петербург, 2002, 608 с.
7. Антонов А.С. Введение в параллельное программирование (методическое пособие). М.: НИВЦ МГУ, 2002, 69 с. [HTML] (http://mvs.icc.ru/documentation/msu/antonov.pdf)

Волканов Д.Ю., Зорин Д.А.

Исследование применимости моделей оценки надёжности для разработки программного обеспечения с открытым исходным кодом

1. Введение

В настоящее время в мире разрабатывается большое количество программных продуктов с открытым исходным кодом (Open Source). В таких проектах участвует большое количество разработчиков, добавляющих свои фрагменты кода в общий репозиторий (хранилище, где находятся различные версии файлов, из которых состоит программа).

Проектом будем считать одну или несколько программ, созданием которых управляет одна команда разработчиков. Большой проект может состоять из подпроектов, реализующих отдельные части, работающие независимо друг от друга. Подпроекты могут состоять из компонент, которые выполняют определённый набор функций, и не используются отдельно. В проекте участвуют программисты, работающие над одним или несколькими подпроектами, тестировщики, и руководители проекта. Одна из основных задач руководителя проекта — определить, когда программу можно предоставлять конечным пользователям.

Основные особенности проектов с открытым кодом следующие:

- Команда тестировщиков непостоянна и работает не в установленные часы, а по мере своих возможностей. Отчёты об ошибках могут поступать в любое время суток (из-за того, что отправители могут находиться в разных городах мира).
- Не гарантируется, что количество времени и усилий, затраченных на тестирование, равномерно.
- Параллельно с тестированием продолжается разработка, поэтому количество ошибок может увеличиться.

В результате, перед руководителями проекта стоят следующие задачи:

- Оценить количество ошибок, оставшихся в проекте, и их критичность.
- Оценить количество ошибок, оставшихся в выбранном компоненте, и их критичность.
- Оценить время, когда текущая версия станет стабильной, то есть количество ошибок и их критичность будут меньше заданного порога.

Для решения этих задач можно использовать два подхода: первый основан на изучении исходного кода программы, второй — на

использовании моделей оценки надёжности (МОН) (англ. software reliability growth model, SRGM) — на данных, полученных при тестировании [1]. Далее в статье подробно рассматривается второй подход.

Основным результатом расчётов в этом случае является *функция количества ошибок* или *функция количества (*англ. mean function) — неубывающая функция времени, показывающая количество ошибок, найденных к заданному моменту времени. При получении этой функции возникают задачи, описанные следующем разделе.

2. Постановка задачи

Введем следующие обозначения:

- *f(t)* – количество ошибок, найденных к заданному моменту: данные, полученные из статистики тестирования.
- *m(t)* – функция количества ошибок, полученная при применении какой-либо модели.
- $m(t)_x$ - функция количества ошибок, полученная в момент времени *x* (на основании статистики до момента *x*).

Изначально задана статистика тестирования: множество ошибок, для каждой из которых обязательно задано время обнаружения и компонент, где она была обнаружена.

Для получения функции *m(t)* из исходных данных результатов тестирования требуется решить следующие задачи:

1. Определить период времени, через который необходимо строить новую функцию *m(t)* для получения более точных оценок, учитывающих вновь поступившие данные.
2. Выбрать наиболее подходящую к данному проекту МОН из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

В качестве критериев качества моделей предлагается использовать следующие:

- Сходимость: возможность построения *m(t)* по исходным данным результатов тестирования и выбранной МОН.
- Точность: среднеквадратичное отклонение полученной функции количества ошибок от реальной: $acc(t) = \frac{\sum_{t_i} \frac{m(t_i)}{f(t_i)}}{t} * 100\%$

- Устойчивость: оценка в текущий период мало отличается от оценки за предыдущий период:

$$st(t) = \left| \frac{m(\infty)_{t_n}}{m(\infty)_{t_{n-1}}} * 100\% - 100 \right| < k, k = 10$$

Данная работа посвящена решению задач 1 - 3 для некоторых конкретных проектов с открытым исходным кодом.

3. Классификация моделей оценки надёжности

Суть МОН заключается в том, чтобы построить математическую модель, описывающую процесс обнаружения ошибок. Для этого применяются различные вероятностные модели случайных процессов, при этом основной случайной величиной, распределение вероятностей которой необходимо найти, является либо количество ошибок в некоторый момент или за определённый период, либо время между двумя ошибками. Выбрав общий вид используемой модели (например, неоднородный пуассоновский процесс или цепь Маркова), можно различными статистическими методами оценить его параметры и в результате получить функцию, показывающую наиболее вероятное количество ошибок в некоторый момент, либо оценку для времени до обнаружения следующей ошибки. После этого можно сравнивать функции, полученные в результате применения различных моделей, между собой и выбирать наилучшую. На основании выбранной модели можно строить оценки различных величин, например, ожидаемое количество ошибок в ближайший месяц или общее количество оставшихся ошибок, на основании которых можно принимать решение о готовности программы к выпуску.

Условно можно разделить все модели на три типа:

1. Марковские. В марковских моделях используется предположение, что количество оставшихся ошибок зависит от текущего состояния системы и не зависит от прошлых состояний [1].
2. Пуассоновские. Предполагается, что обнаружение ошибок можно описать моделью «Поток редких событий» [2], если предположить, что ошибки независимы, обнаруживаются по одной, и известен характер изменения частоты их обнаружения.
3. Байесовские. Время между ошибками считается случайной величиной с показательным распределением, но его параметры зависят от времени появления предыдущих ошибок, при этом оценивается только время до ближайшей ошибки, но не общее количество [3].

Возможны также комбинации этих типов, например, в марковскую модель добавляется байесовская модификация.

Применение МОН к проприетарному программному обеспечению достаточно хорошо исследовано (например, в работах [4], [5], [6]). Проприетарное программное обеспечение в большей степени, чем открытое, удовлетворяет указанным выше ограничениям, так как команда разработчиков и тестировщиков централизованная, и поэтому процесс разработки и тестирования является более равномерным. Кроме того, МОН изначально создавались без учёта возможности применения к проектам с открытым исходным кодом, так как в то время (70-е годы) современный способ разработки проектов с открытым кодом ещё не существовал. Поэтому применимость МОН к открытому программному обеспечению требует дополнительного исследования.

В данном исследовании были проанализированы наиболее подробно описанные в литературе и часто используемые модели оценки надежности:

- Goel-Okumoto (GO) [1]
- S-Shaped (SS) [1]
- Logarithmic (Log) [5]
- Jelinski-Moranda (JM) [4]
- Littlewood-Verrall (LV) [7]

Основной вопрос, появляющийся при использовании SRGM — какую модель использовать в том или ином проекте. Однозначный ответ на этот вопрос найти в общем случае достаточно трудно, более того, нельзя гарантировать, что для одного и того же проекта в течение всего времени разработки будет верна одна модель. Перечислим основные факторы, из-за которых МОН могут работать неточно:

- Малое количество данных. Причиной может быть, например, то, что данные тестирования группируются по неделям или месяцам. Из-за малого количества данных точность оценки параметров снижается.
- Неравномерность тестирования, особенно в сочетании с малым количеством данных
- Стандартные ограничения моделей, то есть наиболее часто встречающиеся допущения о следующих свойствах проекта [7].:
  1. Равномерность: тестирование равномерно, то есть ПО тестируется одинаково до и после построения оценок МОН.
  2. Однородность: все ошибки (по крайней мере, в рамках анализируемой группы) равноценны и вероятность их обнаружения одинакова.
  3. Независимость: ошибки не зависят друг от друга
  4. Идеальность: после обнаружения ошибка немедленно исправляется, и при этом новых ошибок не появляется.
  5. Бесконечность: общее число ошибок считается бесконсчным, нет возможности получить его оценку напрямую.

Чтобы можно было решить, какую из моделей выбрать, необходимы критерии выбора. В следующем разделе будет предложен алгоритм выбора модели.

4.Алгоритм выбора модели оценки надёжности

В этом разделе предложен алгоритм, позволяющий сравнивать несколько заданных МОН[8]. Алгоритм состоит из следующих шагов:

1. На первом шаге рассчитываются параметры для нескольких моделей (методом максимального правдоподобия). После первого шага остаются только те модели, для которых метод сходится. Сходимость может отсутствовать из-за того, что модель в принципе не подходит (например, взята S-изогнутая кривая, а в реальности она строго выпуклая)
2. Исследуется точность моделей. Для этого применяется любой из существующих методов (минимизация среднеквадратичного отклонения, критерий хи-квадрат и другие). В программе, использовавшейся для исследований, рассчитывается среднеквадратичное отклонение, так как этот способ прост и универсален.
3. Проверка на устойчивость (по формулам из раздела 2).
4. Если на предыдущем шаге не оказалось ни одной устойчивой модели, то необходимо продолжить тестирование и сбор данных ещё на один отчётный период. Если же есть устойчивые модели, то на основе их можно уже решать, является ли текущая версия стабильной.

Описанный алгоритм носит эмпирический характер: модель выбирается после некоторого периода, в течение которого производятся различные промежуточные расчёты, тогда как изначально по свойствам проекта никаких предположений не делается. Для того, чтобы проверить работоспособность алгоритма и адекватность выбранных характеристик точности, необходимо испытать модели и алгоритм на реальных данных. Для этих целей было создано программное средство, позволяющее делать расчёты, и в следующем разделе показаны результаты проведённых с его помощью экспериментов.

5. Экспериментальное исследование

5.1 Цель исследования

Задача исследования — на данных тестирования реальных проектов исследовать различные описанные в литературе модели, и получить решение для задач 1-3 из раздела 2:

1. Определить период времени, через который необходимо строить новую функцию *m(t)* для получения более точных оценок, учитывающих вновь поступившие данные.

2. Выбрать наиболее подходящую к исследуемому проекту МОН из нескольких возможных.
3. Оценить, какой объем исходных данных необходим для того, чтобы модели давали результаты с точностью не ниже заданной.

5.2 Выбор проектов для исследования

Проекты выбирались исходя из следующих требований:

- Большой объем программы (сотни тысяч строк)
- Длительное время тестирования, обширная база данных о тестировании (несколько лет).
- Большое количество разработчиков (несколько сотен)
- Состоит из отдельных компонент.

Для практического исследования были выбраны несколько проектов с открытым исходным кодом. Так как модуль, собирающий статистику, ориентирован на систему Bugzilla [9], выбирались проекты, использующие эту систему для хранения статистики тестирования. В качестве основного был взят проект Eclipse [10], в качестве дополнительных — Mozilla [11] и Linux Kernel [12].

Проект Eclipse был выбран в качестве основного, так как содержит самую большую базу (около 260 000 отчётов об ошибках) с наиболее подробной статистикой: указаны дата, модуль, фатальность ошибки, текущий статус для каждой ошибки.

5.3 Методика экспериментального исследования.

1. По выбранным проектам из системы отслеживания ошибок загружается статистика тестирования.
2. Для этих проектов и, выборочно, их компонент и подмножеств ошибок одинаковой критичности, получены следующие данные:
   - Оценка количества ошибок на ближайший период и сравнение этой оценки с реальными данными (по периодам за некоторое время).
   - Точность и устойчивость рассматриваемых сходящихся моделей, в соответствии с алгоритмом из раздела 4.
3. Используя эти данные, с помощью алгоритма из раздела 4 получается ответ на задачи 2 и 3. Оценивая точность полученных результатов на разных промежутках времени между применениями алгоритма выбора модели, можно получить ответ на задачу 1.

5.4 Эксперименты

Полученные в результате исследования данные показывают, что точность и устойчивость, начиная приблизительно с 10-15 месяца, стабильно отклоняются от 100 не более, чем на 10 процентов (рисунок 1-2), что можно считать приемлемым. Из этого можно сделать вывод о

том, что в целом МОН применимы к проектам с открытым исходным кодом, несмотря на децентрализованность тестирования и совмещения тестирования с разработкой.

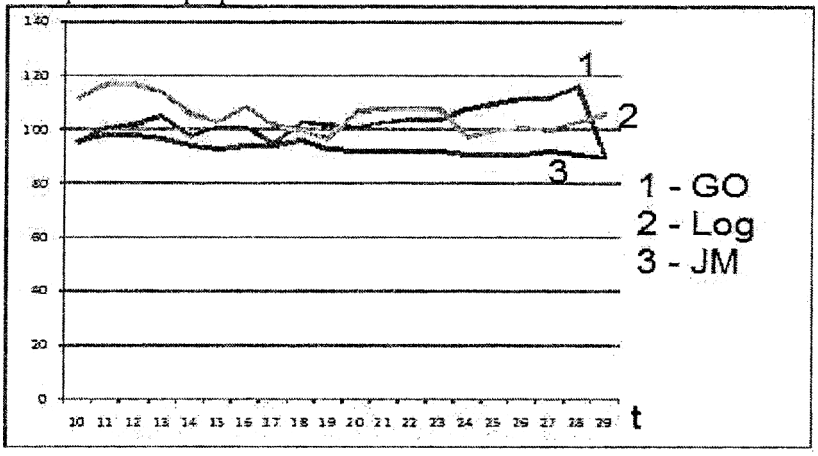


Рис. 1. Подпроект JDT – функция ac(t)

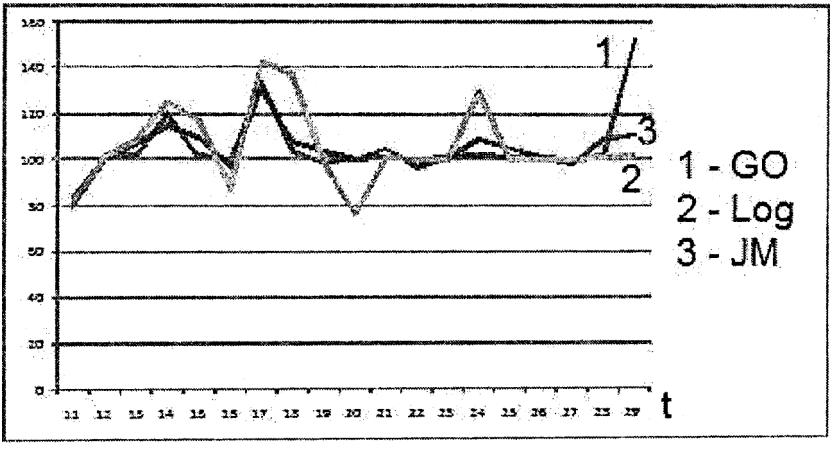


Рис. 2. Подпроект JDT – функция st(t)

Из экспериментов также сделан вывод, что наиболее приемлемым периодом для перерасчёта является один месяц. При сравнении данных, полученных при разной частоте пересчёта, обнаруживается, что близость оценки к реальным данным при разбиении по месяцам значительно выше, чем при разбиении по неделям (в среднем 10-15% против 30-40%). При разбиении по неделям расхождения оценок и результатах более явные. При разбиении по годам же данных слишком

мало и уменьшения количества ошибок не прослеживается. Такие результаты можно объяснить тем, что для расчёта по годам / полугодиям нужно слишком много данных, а при расчёте каждую неделю слишком большое влияние оказывают факторы, не связанные с тестированием непосредственно — нерабочие дни, релизы новых версий и так далее. Кроме того, так как при перерасчёте по годам виден почти линейный рост, можно отметить, что за достаточно большой период набирается достаточно нового кода, содержащего новые ошибки (по этой же причине предположение о бесконечном количестве ошибок в коде не является существенным ограничением в данном случае). При расчёте же по неделям точность сильно понижается. Тем не менее, вопрос о более точном определении расчётного периода остается открытым, возможно, стоит найти формулу или алгоритм для определения оптимальной величины расчётного периода, если имеется достаточно большой объем статистики.

При этом также можно отметить, что полученные оценки достаточно точны в течение более длительного времени, чем один месяц, вплоть до 3-4 месяцев (рисунок 3).

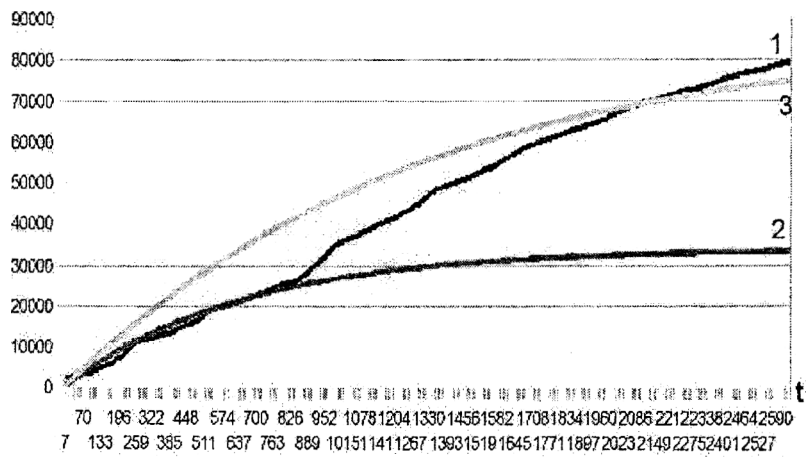


Рис. 3. 1 – функция f(t), 2 – функция $m(t)_{100}$, 3 — функция $m(t)_{300}$.

Видно также, что приемлемая точность аппроксимаций и оценок достигается примерно к 10-15 месяцу после начала тестирования, в начале же она очень низкая. Это связано, в первую очередь с особенностями самих моделей — до тех пор, пока тестирование не дойдет до того момента, на котором кривая начинает приближаться к асимптоте, очевидно, что стремление к точности аппроксимации будет снижать точность оценок. Кроме того, в случае с Eclipse у данного

проекта есть особенность — в начале в Bugzilla, по всей видимости загрузили статистику за некоторый период, поставив близкие даты.

Что касается сравнения моделей между собой, то показатели моделей Goel-Okumoto, Jelinski-Moranda и Logarithmic в целом довольно близкие. Проведенные расчеты свидетельствуют, что спустя некоторое время (те же 10-15 месяцев) разница между оценками по этим моделям оказывается не выше 5%. Минусом GO модели является то, что она иногда не сходится — постоянно применять только ее не всегда возможно. Модель JM можно считать наиболее устойчивой, но не самой точной, кроме того она делает скорее заниженные оценки, тогда как из практических соображений более приемлемыми следует считать несколько завышенные.

Наиболее подходящей кривой для проектов с открытым исходным кодом, как можно было предположить, является S-изогнутая кривая, что хорошо видно на примерах статистики по отдельным модулям. Причиной этого является то, что у каждого небольшого модуля имеется относительно небольшая команда разработчиков, которые другими модулями почти не занимаются. Как следствие, после обнаружения большого количества ошибок в начале качество тестирования начинает повышаться. В более глобальном масштабе, на уровне целого проекта уровня JDT или Platform (количество ошибок за весь период имеет порядок нескольких десятков тысяч) S-изогнутая модель работает плохо, что можно объяснить неравномерностью разработки по разным модулям.

5.5 Выводы

В результате проведенного исследования были сделаны следующие выводы:

- Лучше всего строить новые оценки каждый месяц.
- Высокая точность и устойчивость достигаются через 10-15 месяцев после начала тестирования.
- Наиболее подходящая МОН для проектов с открытым исходным кодом – S-Shaped, при этом ее лучше применять не к целым проектам, а к подпроектам.

6. Заключение

В рамках данной работы получены следующие результаты:

- Предложен алгоритм сравнения МОН.
- Реализовано программное средство для сравнения моделей и выбора наиболее подходящей.
- Показана общая эффективность МОН применительно к выбранным проектам с открытым кодом.
- Выбран наиболее подходящий промежуток времени для разбиения данных — один месяц.

Сформулированы следующие гипотезы о применении МОН к проектам с открытым исходным кодом:

- Приемлемая точность и устойчивость достигается через 10-15 месяцев после начала тестирования.
- Наиболее подходящей для проектов с открытым исходным кодом является S-изогнутая пуассоновская модель.

В качестве дальнейшего развития данного подхода можно выделить следующие направления:

- Автоматизация выбора наилучшего периода для группировки данных (с возможностью выбора произвольного периода, а не только «круглого»).
- Проверка указанных выше гипотез статистически, например с помощью критерия Пирсона. Для этого необходимо получить результаты для большего количества проектов.
- Определение периода, на котором оценки верны с точностью не ниже заданной.

Литература

1. M.Lyu Software Reliability Engineering: A Roadmap // 2007 Future of Software Engineering. Washington, DC, USA, 2007. p. 153 — 170.
2. Ширяев А. Н. Вероятность. М.: МЦНМО, 2004.
3. R.Al-Ekram Software Reliability Growth Modeling and Prediction. Waterloo: Dept. of Electrical and Computer Engineering, University of Waterloo, 2002
4. S.Alam Software Reliability Using Markov Chain Usage Model. Dhaka, Bangladesh: Depatment of Computer Science and engineering Bangladesh University of Engineering and Technology, 2005
5. A.Wood Software Reliability Growth Models. Technical Report, 1996
6. H.Umeda The development of a program drawing reliability growth curves and implementing reliability growth models. Tottori University of Natural Sciences, Department of Infromation Systems, 2008
7. M.Limnios, N.Nikulin Recent Advances in Reliability Theory. Birkhauser, 2000
8. C.Stringfellow An Empirical Method for Selecting Software Reliability Growth Models // Empirical Software Engineering 2002. 7. №4. p. 319 — 343.
9. Mozilla project. Bugzilla bug tracker http://www.bugzilla.org/about/
10. Eclipse project. http://bugs.eclipse.org/bugs/
11. Mozilla Project. https://bugzilla.mozilla.org/
12. Linux Kernel project. http://bugzilla.kernel.org/

Леонов М.В., Корнев Д.Б., Страхов В.Н.

Пакет KSWeb: новый инструмент разработки мобильных информационных систем

Введение

Под мобильными информационными системами далее будем понимать информационные системы, не требующие инсталляции на жестких дисках компьютера. Для разработки таких систем удобно использовать так называемые «web-серверные сборки», или web-серверные пакеты, в состав которых, кроме собственно Web-сервера, включают интерпретатор скриптового языка, например PHP, Perl, а также СУБД, в роли которой чаще всего оказывается MySQL.

Примерами таких пакетов являются microweb [1], nanoweb [3], Denwer[6], xampp [5], MoWes [2]. Причиной многообразия и активизации разработок в том направлении, по-видимому, являются, с одной стороны, необходимость удобного тестирования Интернет-приложений с базами данных на локальном компьютере, а с другой – снижение популярности таких коммерческих систем, как FoxPro или Access.

В процессе выполнения совместных проектов по автоматизации научных исследований в рамках сотрудничества между Елецким Государственным Университетом имени И.А. Бунина и факультетом ВМК МГУ возникла потребность в собственной Web-серверной сборке, которая обеспечивала бы функционирование разрабатываемых информационных систем с Web-интерфейсом.

1. Характеристики и особенности пакета KSWeb

В пакет входят - Web-сервер KSWeb, написанный на языке Java в среде разработки NetBeans 6; СУБД MySQL, интерпретатор PHP, PHPMyadmin (веб-ориентированный администратор баз данных MySQL), мобильная версия браузера FireFox Portable

Основными требованиями к пакету были: возможность использования его вместе с соответствующей информационной системой непосредственно на флэш-носителе или на CD, без необходимости инсталляции на жестком диске компьютера.

Для удобства настройки сервера в состав пакета входит конфигурационный файл, в котором можно задать следующие опции:

запускать браузер в момент старта сервера или нет;

использовать ли браузер, встроенный в пакет, либо запускать браузер, зарегистрированный в системе браузером по умолчанию;

указать адрес для адресной строки браузера;

указать порт сервера(числовое значение от 0 до 65535);

указать корневой каталог для файла INDEX_HTML_FILE и всех остальных;

указать имя файла с сообщением об ошибке 404(файл не найден).

Для получения количественных и качественных характеристик было проведено тестирование нашего Web-сервера с помощью утилиты Apache Benchmark. В этих тестах, кроме KSWeb, тестировались Microweb (пакет версии 2.12), Denwer (пакет дата выпуска 2008-01-13, с apache 2.2.4; php 5.2.4; MySQL 5.0.45; phpMyAdmin 2.6.1), xampp (пакет v1.6.4), и TJWS (сервер v1.51), также написанный на языке Java.

Тесты проводились на платформе со следующими характеристиками: AMD Athlon 3000+; 1024mb RAM; GeForce 7600GS 128mb; Seagate Baracuda 160gb; Windows XP x86 SP3

Первый тест заключался в запросе 100 раз файла размером 90319 байт по технологии Keep-Alive, то есть единожды подключившись к серверу, производилось100 запросов, после чего выполнялось разъединение.

1 место	Xampp	Результат: 427мс	
2 место	Denwer	Результат: 448мс	
3 место	KSWeb	Результат: 458мс	
4 место	TJWS	Результат: 458мс	
5 место	Microweb	Результат: N/A	причина: microweb не поддерживает Keep-Alive

Второй тест заключался в запросе 100 раз файла размером 90319 байт без использования технологии Keep-Alive, то есть с подключением к серверу при КАЖДОМ из 100 запросов и таким же количеством отключений. Как и следовало ожидать, наблюдалось увеличение времени относительно предыдущего теста у всех испытуемых.

1 место	Xampp	результат: 510мс
2 место	Denwer	результат: 521мс
3 место	TJWS	результат: 609мс
4 место	KSWeb	результат: 640мс
5 место	Microweb	результат: 916мс

Третий тест заключался в запросе 100 раз PHP- скрипта. Содержимое контента после обработки одного скрипта составляло 157002 байт. Запросы производились без использования технологии Keep-Alive, то есть при КАЖДОМ из 100 запросов выполнялось подключение к серверу с последующем отключением.

Текст скрипта:

```
<html>
    <head> <meta http-equiv="Content-Type" content="text/html;
     charset=windows-1251"></head>
    <body>
        <form action = "multipart2.php" enctype="multipart/form-
     data" method="post">
                        <input type = "submit">
                        <?
                                $i=0;
                                for($i;$i<=2500;$i++){
                                        echo "<input type=text
     name=name$i value=name$i size=30 name=$i>";
                                }
                        ?>
        </form>
    </body>
</html>
```

1 место	Xampp	результат: 1614мс	
2 место	Denwer	результат: 1630мс	
3 место	KSWeb	результат: 8599мс	
4 место	Microweb	результат: 26558мс	
5 место	TJWS	результат: N/A	причина: TJWS не поддерживает php-скрипты

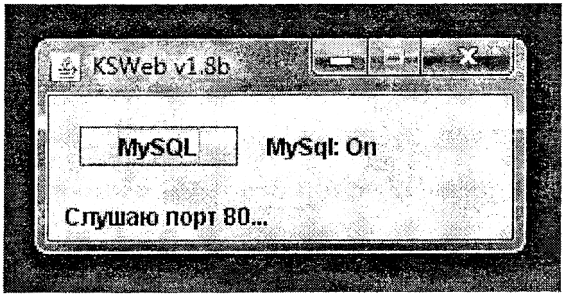
Краткий вывод: из результатов проведённых тестов видно, что лидером из всех участников по скорости является xampp на базе сервера apache (PHP подключается в виде модуля). От xampp-а не отстаёт и denwer. Это не удивительно, ведь в основе этих двух пакетов лежит один и тот же сервер apache. Из таблицы видно, что пакет KSWeb показывает средние результаты по быстродействию.

Относительно большой проигрыш лидерам у нашего испытуемого в тесте с участием PHP. Это объясняется тем, что PHP работает через интерфейс CGI. Microweb не поддерживает Keep-Alive и это его очевидный минус. В тесте PHP он занял последнее место, причем со значительным отставанием от других испытуемых (более чем в 2раза).

1 место	Xampp
2 место	Denwer
3 место	KSWeb
4 место	Microweb
5 место	TJWS

Данные тестирования характеризуют скорость работы с пакетами. Заметим, быстродействие не было главной целью разработки нашего пакета. Его работоспособность и возможности использования были

успешно проверены на таких ресурсоемких веб-приложениях, как Joomla, Drupal, Moodle.



Для удобства контроля за состоянием сервера и сервера СУБД при запуске сервера выводится специальное окно (см. Рис.1).

Рис. 1.

2. Применение пакета: электронная мемориальная имидж-картотека членов общества МОИП

Первым примером реального использования нашего сервера было его использование в проекте создания электронной мемориальной имидж –картотеки членов Московского Общества Испытателей Природы (МОИП) – старейшего научного общества России. Суть проекта следующая. В настоящее время в библиотеке МОИП существует картотека его членов, которую иногда называют дореволюционной, хотя она содержит данные по членам Общества в период с 1805 по примерно 1990 годы. Карточки – рукописные, содержат записи и пометки нескольких поколений библиотекарей и технических секретарей. Данные этой картотеки представляют большой научный интерес для изучения истории естествознания и самого Общества. Для расшифровки, сохранения и вовлечения в научный оборот этих данных было решено создать так называемую «имидж-картотеку» с возможностью пополнения текстовой части базы данных персоналий членов МОИП. Для управления этой «имидж-картотекой» необходимо было разработать специализированное программное обеспечение с Web-интерфейсом, которое позволило бы использовать эту базу данных как в индивидуальном режиме, так впоследствии и через Интернет.

Для этой цели как нельзя лучше подходит идея мобильной информационной системы: такую систему можно использовать в читальном зале библиотеки, в архиве, дома, в командировке и т.д. Применение собственного сервера KSWeb вполне оправдало себя. В настоящее время система управления этой электронной картотекой находится в опытной эксплуатации.

Заключение

Представленный Web-серверный пакет предназначен в основном для разработки мобильных информационных систем на основе СУБД MySQL и PHP, используемых на флэш-носителях и CD. Пакет также удобно использовать для отладки информационных систем указанного класса, предназначенных для последующего размещения на Интернет-сайтах.

В настоящее время пакет KSWeb используется для отладки двух систем автоматизации научных исследований в историко-архивном проекте по созданию Мемориальной электронной картотеки Московского Общества Испытателей Природы и для историко-библиографического проекта для обеспечения доступа к данным дореволюционного Журнала Министерства Народного Просвещения (1834–1917 гг.).

Кроме того, представленный пакет успешно используется в учебном процессе для выполнения практических заданий на спецсеминарах по Web-технологиям для бакалавриата и вечернего спецотделения факультета ВМК МГУ.

Ссылки

1. http://www.indigostar.com/microweb.htm
2. http://www.chsoftware.net/en/useware/mowes/mowes.htm
3. http://nanoweb.si.kz/
4. http://tjws.sourceforge.net/
5. http://www.apachefriends.org/en/xampp.html
6. http://www.denwer.ru/

Шумкин Г.Н.

Математическое моделирование молекулярных переключателей

Работа поддержана грантом РФФИ 08-07-12081 и 09-07-12068

1. Введение

Создание молекулярных устройств, которые можно переключать между состояниями с высоким и низким сопротивлениями, является одним из важнейших направлений нанотехнологий. Так как миниатюризация этих технологий продолжает развиваться, фундаментальная проблема идентифицирования и понимания наименьших физических систем, которые способны к переключательному поведению, привлекает растущий интерес [1-5]. Последние эксперименты в научно-исследовательской лаборатории IBM в Цюрихе связаны с открытием индуцированной током изомеризации молекулы нафталоцианина и наблюдением переключения проводимости молекулы, не приводящее к изменению геометрической формы молекулы [4].

В последнее время вычисления из "первых принципов" с использованием супер-ЭВМ начали широко использовать для моделирования молекулярных переключателей [1-4]. Данная статья посвящена вычисления высоты барьера реакции изомеризации в молекуле нафталоцианина с целью оптимизации процесса переключения.

2. Основные уравнения и постановка задачи

При построении моделей молекулярной динамики из "первых принципов" исходят из уравнения Шредингера для молекулярной системы, в которые включены только кулоновские взаимодействия и отсутствуют эмпирические, или подгоночные, параметры.

В исследовании используется математическая модель молекулярной динамики Кар-Парринелло [6] на основе приближения Борна-Оппенгеймера. В приближении Борна-Оппенгеймера считается, что движение массивных ядер может быть описано в классическом приближении. Силы, действующие на ядра со стороны электронов рассчитываются на основе решения уравнения Шредингера для электронов в основном состоянии. Модель состоит в учете электронной структуры при решении уравнений молекулярной динамики для ядер. Электронная структура получается для каждой молекулы решением стационарного уравнения Шредингера для каждого шага классической молекулярной динамики ядер, т.е. положения ядер являются

параметрами для электронного уравнения Шредингера. В результате уравнения молекулярной динамики Борна-Оппенгеймера записываются следующим образом:

$$M_I \ddot{\mathbf{R}}_I(t) = -\frac{\partial E_0}{\partial \mathbf{R}_I}\{\langle\psi_0|H_e|\psi_0\rangle\} \quad (1)$$

$$H_e\psi_0 = E_0\psi_0 \quad (2)$$

Электронный Гамильтониан $\mathrm{H_e}$ записывается в виде:

$$H_e(\{\mathbf{r}_i\},\{\mathbf{R}_I\}) = -\sum_i \frac{\hbar^2}{2m_e}\nabla_i^2 + \sum_i\sum_j \frac{e^2}{|\mathbf{r}_i-\mathbf{r}_j|} \quad (3)$$
$$-\sum_I\sum_i \frac{e^2 Z_I}{|\mathbf{R}_I-\mathbf{r}_i|} + \sum_I\sum_J \frac{e^2 Z_I Z_J}{|\mathbf{R}_I-\mathbf{R}_J|}$$

где $\{\mathbf{r}_i\}$ это электронные и $\{\mathbf{R}_I\}$ ядерные степени свободы.

Здесь электронные состояния рассматриваются в основном состоянии, о чем говорит индекс ноль у волновой функции.

$$E_0 = \min_{\psi_0}\{\langle\psi_0|H_e|\psi_0\rangle\} \quad (4)$$

При расчете уравнения для электронов (2) будет использоваться квантовомеханическая формулировка, основанная на уравнениях функционала плотности "DFT" [7-9].

Модель состоит в поиске решения уравнений Кона-Шема ("KS") для электронов

$$(-\frac{1}{2}\Delta + U_{eff}(\mathbf{r},\rho))\psi_i(\mathbf{r}) = \epsilon_i\psi_i(\mathbf{r}) \quad (5)$$

Здесь ψ_i есть набор одночастичных электронных волновых функций, ϵ_i - их энергии и $U_{eff}(\mathbf{r},\rho))$ эффективный потенциал, зависящий от электронной плотности ρ. Электронная плотность ρ(r)

системы как центральная переменная в методе DFT определяется по заполненным KS орбиталям:

$$\rho(\mathrm{r}) = \sum_{i}^{occ} |\psi_{i,\sigma}(\mathrm{r})|^2 \quad (6)$$

и затем определяется функционал энергии Кона-Шэма E_{KS}.

Наиболее успешный метод решения задачи на собственные значения электронного (2) уравнения в настоящее время является метод Кар-Парринелло. Данный метод реализован в квантовомеханическом коде CPMD [10-11], который использовался в настоящем исследовании.

3. Нахождения пути химической реакции изомеризации на поверхности потенциальной энергии молекулярной системы

На рис.1 показана рассчитанная оптимизированная по энергии молекуле нафталоцианина (черные атомы соответствуют атомам азота, серые - атомам углерода, белые – атомам водорода). В эксперименте под действием потенциала иглы сканирующего туннельного микроскопа два атома водорода в центральной полости молекулы нафталоцианина переходят к другой паре атомов азота, что соответствует формальному повороту молекулы на 90 градусов.

Исследование пути химической реакции равносильно поиску седловых точек на многомерной поверхности потенциальной энергии, зависящей от положений всех ядер. Т.к. поверхность многомерная и использование традиционных методов поиска экстремума невозможно, то необходимо использовать внутренние координаты реакции. Внутренняя координата реакции (IRC) [12] определяется как наикратчайший путь на поверхности потенциальной энергии во взвешенных массами декартовых координатах, которые связывают (химическое) состояние реагента с состоянием продукта реакции. Вычисляются IRC для определения переходных состояний. Для изучения реакции водородной таутомеризации, проходящей в молекуле нафталоцианина, была предложена коллективная переменная, которая учитывает координаты двух атомов водорода и двух атомов азота в центральной полости молекулы. Определим координационную переменную в виде:

$$s_1(H_1, H_2, N_1, N_2) = \frac{1}{2}\left\{ \frac{1-(\frac{R_{H_1N_1}}{R_0})^6}{1-(\frac{R_{H_1N_1}}{R_0})^{12}} + \frac{1-(\frac{R_{H_1N_2}}{R_0})^6}{1-(\frac{R_{H_1N_2}}{R_0})^{12}} + \frac{1-(\frac{R_{H_2N_1}}{R_0})^6}{1-(\frac{R_{H_2N_1}}{R_0})^{12}} + \frac{1-(\frac{R_{H_2N_2}}{R_0})^6}{1-(\frac{R_{H_2N_2}}{R_0})^{12}} \right\}$$

Здесь через $R_{H_iN_j}$ обозначены расстояния между i-м атомом водорода и j-м атомом азота. Расстояние R_0 есть равновесное расстояние между атомом водорода и атомом азота, равное 1.05A. Использовался следующий выбор координационной переменной:

$$s = s_1(H_1, H_2, N_1, N_2) - s_1(H_1, H_2, N_3, N_4)$$

Выбор такой формы координационной переменной оказывается симметричным по отношению к пространственному повороту.

4. Определение высоты потенциального барьера химической реакции методом "метадинамика".

Центральным местом в моделировании молекулярного переключения является расчет барьера в свободной энергии Гиббса между двумя изомерными равновесными состояниями.

Вычисление пути реакции, который проходит через морщинистый энергетический ландшафт, есть важный, но трудный шаг в понимании механизмов химических реакций.

Длительность поиска очевидна, так как в молекулярной динамике системы реакция представляет редкое событие. Предложено несколько алгоритмов поиска путей реакции для сокращения времени поиска: статистические методы анализа взвешенных гистограмм [13], алгоритм зонтика [14], метод волновых пакетов, метод "метадинамики" [15], [16]. Рассмотрим здесь метод "метадинамики" [15], [16], реализованный в используемом коде CPMD. Различные рецепты выбора параметров в "метадинамике" и предварительного использования статистических методов поиска представлены в работе [16].

Целью метода метадинамики является эффективный анализ поверхности свободной энергии $F(s)$ как функции ограниченного набора коллективных переменных s (длины связей, углы, координационные числа). Свободная энергия обычно записывается в виде:

$$F(\mathrm{s}) = -k_B T ln[Z(\mathrm{s})] \quad (7)$$

где k_B есть константа Больцмана, T - абсолютная температура, и Z есть функция распределения

$$Z(\mathrm{s}) = \frac{1}{Z}\int d\mathrm{r} \cdot e^{\frac{H(\mathrm{r})}{k_B T}} \delta(\mathrm{s}(\mathrm{r}) - \mathrm{s}') \quad (8)$$

где $H(r)$ - Гамильтониан. Вводится фиктивная частица с координатой s_α для каждой коллективной переменной S_α и используется техника расширенного Лагранжиана. Структура расширенного Лагранжиана выглядит следующим образом:

$$L = L_{CPMD} + \sum_\alpha \frac{1}{2}\mu_\alpha \dot{s}_\alpha^2 - \sum_\alpha \frac{1}{2}k_\alpha (S_\alpha(\mathrm{r}) - s_\alpha)^2 - V(\mathrm{s},t) \quad (9)$$

Последний член в (9) есть зависящий от предыстории потенциал:

$$V(\mathrm{s},t) = \sum_{t_i<t} H \cdot exp[-\frac{(\mathrm{s}-s_\mathrm{i})^2}{2(\Delta W)^2}] \quad (10)$$

Здесь $s_i = s(t_i)$. Потенциал V(s,t) описывает медленно растущую многомерную гауссову трубку с осью вдоль траектории. На практике стартуют без добавления холмов, так что V(s,t) = 0 и молекулярная система и фиктивные частицы флуктуируют в одной из потециальных ям (ямы реагента). Флуктуации s_α дают информацию о ширине ямы, из которой выбирается ширина гауссового холма ΔW. "Метадинамика" s_α определяет эффективность, с которой анализируется поверхность свободной энергии. Затем добавляются потенциальные холмы в зависящий от предыстории потенциал V (s,t) с меташагом τ_{MTD}, который на порядок или два больше, чем шаг молекулярной динамики τ_{MD}.

Опишем расчет реакции таутомеризации молекулы нафталоцианина с использованием алгоритма "метадинамики".

Вычисление профиля поверхности свободной энергии вдоль выбранного координационного числа сродни решению стохастического дифференциального уравнения. Поэтому следует определить, что мы понимаем под точностью и сходимостью вычислений. На рис. 2 представлен рассчитанный профиль свободной энергии вдоль координационной переменной. Сходимость метода достигнута путем специального выбора последовательности параметров. По существу, последовательно проводятся расчеты "метадинамики" от ситуации, имевшейся с предыдущими параметрами.

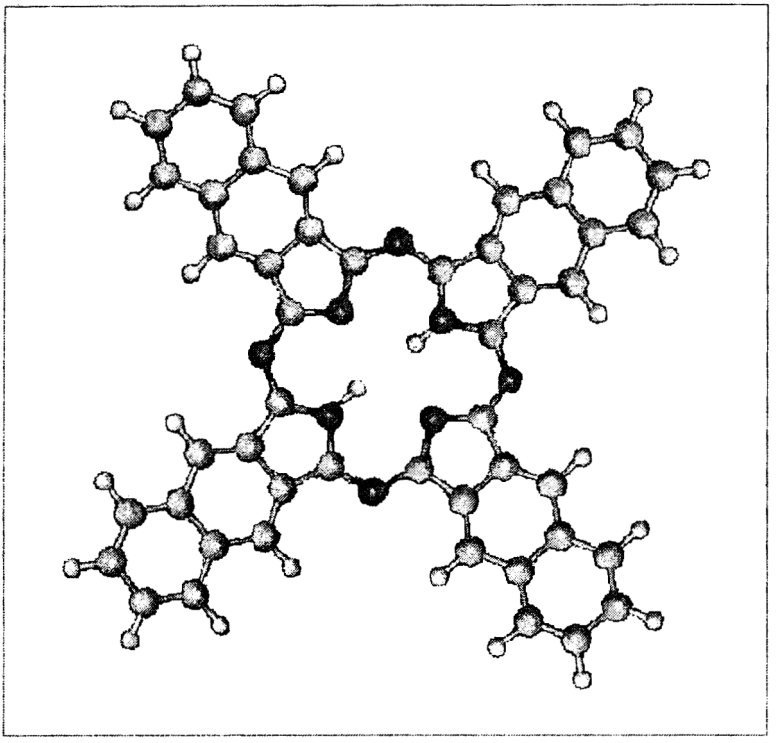


Рис. 1. Геометрия молекулы нафталоцианина

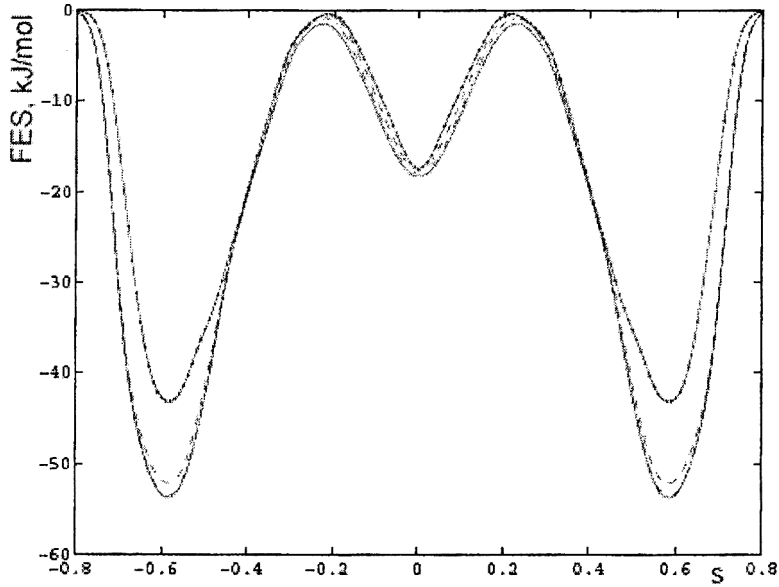


Рис. 2: Сходимость процесса метадинамики для молекулы нафталоцианина. Последовательное восстановление поверхности свободной энергии (FES) в реакции изомеризации. Сплошная линия с точками соответствует расчету с $H=3k_BT$, пунктиром расчету с $H=1.5k_BT$, пунктир-точка линия расчету с $H=0.75k_BT$, сплошная линия расчету с $H=0.375k_BT$

Два равновесных состояния соответствуют значениям координационного параметра s = 0.6 и s = -0.6, значению s = 0.0 отвечает переходное состояние. Расчет состоял из четырех последовательных вариантов - запусков программы с различными параметрами. Первый запуск был выполнен с высокими гауссианами с высотой $H=3k_BT$ и полушириной $\Delta W=0.03$. Заметим, что в этих расчетах полуширина не менялась, а температура соответствовала T=30K. За окончание расчета принималось состояние, когда система заполняла оба минимума. Затем поверх вычисленного потенциала V_G (т.е. начальные конфигурации и скорости были взяты из расчета первоначальной термолизации, после чего проведена повторная термолизация в присутствии потенциала V_G) выполнялся расчет с гауссианами с в два раза меньшей высотой $H = 1.5k_BT$. Таким же образом были выполнены следующие последовательные два расчета с высотой $H=0.75k_BT$ и $H=0.375k_BT$. Шаг метадинамики τ_{MTD} составлял 800 шагов молекулярной динамики τ_{MD}, который составлял 0.1125 фс. Установившееся значение высоты барьера равно 52 кДж/моль при

точности вычислений 1кДж/моль. После установления пути реакции методом "метадинамики" был проведен расчет потенциальной поверхности Борна-Опенгеймера(см рис. 3). В этом случае атом водорода расставляется в различных точках центральной полости молекулы. Вычисленная в этом расчете высота барьера составляет 92 кДж/моль, что почти в 2 раза выше барьера, вычисленного методом "метадинамики".

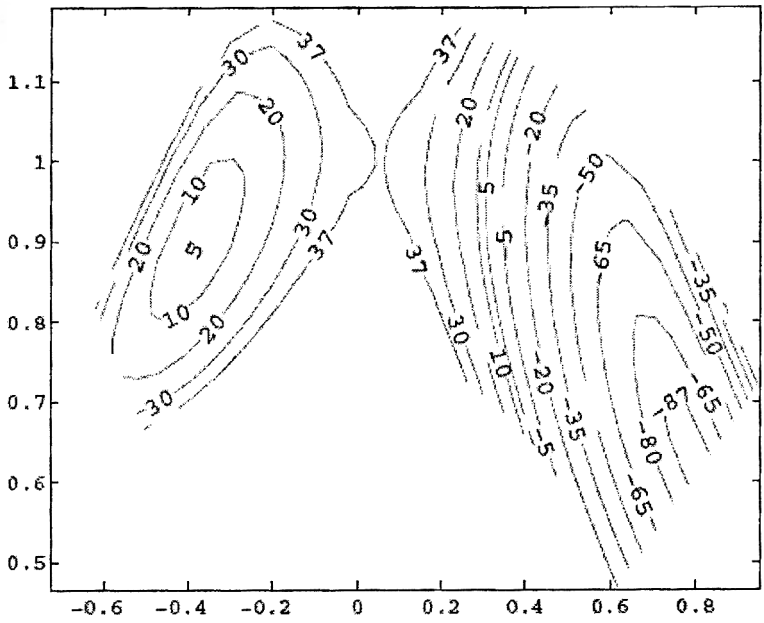


Рис. 3: Линии уровня поверхности Борна-Опенгеймера, рассчитанные для молекулы нафталоцианина, по оси отложены координаты в ангстремах, изолинии даны в кДж/моль

Заключение

В работе приведены результаты моделирования молекулярного переключения с использованием параллельного квантовомеханического кода CPMD на супер-ЭВМ IBM BlueGene/P. В результате была установлена с высокой точностью высота барьера химической реакции, ответственной за переключение в молекуле нафталоцианина, которое наблюдалось в эксперименте [4]. Автор благодарит своего научного руководителя проф. Попова А.М. Работа была выполнена при поддержке грантами РФФИ 08-07-12081 и 09-07-12068.

Литература

1. А.М.Попов.- Вычислительные нанотехнологии .- М. :МАКС Пресс ,2009.-280с.
2. Handbook of Theoretical and Computational Nanotechnology, Third by Michael Rieth and Wolfram Schommers, Karisruhe, Germany, 10-Volume Set, 2006, 8000 p.
3. Eldon G. Emberly and George Kirczenow, The Smallest Molecular Switch .- Phys. Rev.Lett. 91 , № 18, 188301 (2003).
4. Peter Liljeroth, Jascha Repp, Gerhard Meyer.- Current-Induced Hydrogen Tautomerization and Conductance Switching of Naphthalocyanine Molecules .- Science 317 , 31 August, pp. 1203-1206,(2007).
5. Nano and Molecular Electronics, (Handbook) Edited by Sergey Edward Lyshevski, .- CRC Press, Taylor and Francis Group, (2007).
6. Car,R. and Parrinello, M. Unified approach for molecular dynamics and density- functional theory. .- Phys.Rev.Lett. 55, 1985,pp 2471-2474
7. W.Kohn and L.J.Sham .- Physical Review A 140,1133 (1965).
8. W.Kohn and L.J.Sham, Self-consistent equation including exchange and correlation effects, .- Phys.Rev., 140,A1113(1965)
9. W.Kohn, Density Functional and Density Matrix Method Scaling Linearly with the Number of Atoms, .- Phys.Rev.Lett., 76,3168(1996).
10. http://www.zurich.ibm.com/deepcomputing
11. W.Andreoni and A.Curioni New Advances in Chemistry and Materials Science with CPMD and Parallel Computing .- Parallel Computing, 26,819-842(2000)
12. K. Fukui .- Acc. Chem. Res. , 14, 363 (1981).
13. S.Kumar, D.Bouzida, R.H.Swendsen .- J.Comput. Chem. 1992,13,1011
14. J.Kastner, W.Thiel. Bridging the gap between thermodynamic integration and umbrella sampling provides a novel analysis method: "Umbrella Integration" .- J.Chem. Phys., 123,144104(2005).
15. Alessandro Laio, Michele Parrinello, Escaping free-energy, .- PNAS, vol.99, №20,(2002).
16. Bernd Ensing, Alessandro Laio, Michele Parrinello, and Michael L. Klein; A Recipe for the Computation of the Free Energy Barrier and the Lowest Free nergy Path of Concerted Reactions .- J. Phys. Chem. B, 109(14) , 6676-6687 (2005).

Аннотации

Брусенцов Н.П. Отчего классическая логика лишена необходимого следования // Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 6-7.

Необходимое следование трехзначно, с духзначностью классической логики несовместимо. Оно выродилось в материальную импликацию с ее неустранимыми парадоксами. Непарадоксальным следованием обладает аристотелева силлогистика, в двухзначных исчислениях неотображаемая.

Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 20 апреля 2009 г.

Библиогр.: 4 назв.

Владимирова Ю.С. Программная реализация аристотелевой силлогистики. //Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 8-11.

Рассматривается способ адекватного представления содержательного необходимого следования силлогистическими посылками, выраженными суждениями существования. Приводится метод компьютеризации содержательного рассуждения посредством кодирования силлогистических суждений векторами тритов и созданием процедур, осуществляющих силлогистический вывод. Доложено на Ломоносовских чтениях на факультете ВМиК МГУ 20 апреля 2009 г.

Ил.: 4 рис. Библиогр.: 4 назв.

Рябов Г.Г. (НИВЦ МГУ) Алгебраическое представление кубических структур и супервычисления. //Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 12-26.

Предлагаемая статья рассматривает нетрадиционный подход к кодированию и операциям над кубическими структурами, который позволяет объединить алгебраической структурой – полугруппой с единицей (моноидом): возможность подхода к кодированию симплициальных комплексов, действий симметрической группы подстановок (и ее подгрупп) на кубанты и представление случайных троичных n-разрядных слова, как случайных кубантов в n-кубе и множество таких слов как случайный кубический комплекс с привлечением аппарата марковских цепей к анализу эргодических свойств случайных комплексов.

Ил.: 5 рис. Библиогр.: 18 назв.

Королев Л.Н. Оценка вычислительной сложности некоторых алгоритмов построения классификаторов. //Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 27-40.

Предложена методика оценки глобальной вычислительной сложности и временные затраты некоторых алгоритмов прямого перебора, возникающих при решении задач построения классификаторов, кластеров и решателей на компьютерах с массовым параллелизмом. Перебору подлежат параметризованные критерии близости с использованием нейросетей и принципов «генетического программирования». Полученные оценки позволят прикинуть размерности векторных пространств признаков классифицируемых объектов, с которыми справятся современные вычислительные системы.

Библиогр.: 25 назв.

Джосан О.В., Попова Н.Н. Метод сжатия изображений для визуализации на массивно-параллельных вычислительных системах.//Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 41-51.

В работе рассмотрен метод параллельного сжатия изображений и видео для системы визуализации результатов научных вычислений, проводимых на терафлопсных и петафлопсных вычислительных системах.

Ил.: 6 рис. Библиогр.: 8 назв.

Глазкова Е.А., Попова Н.Н. Исследование влияния виртуальных топологий MPI и способа назначения процессов на процессоры на эффективность выполнения гибридной программы на примере системы Blue Gene/P// Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 52-63.

В статье исследуется влияние виртуальных топологий MPI и способа назначения процессов на процессоры на эффективность выполнения модельной гибридной программы - решения разностной задачи Дирихле методом Якоби в трехмерной области. Приведены результаты экспериментов.

Ил.: 10 рис. Библиогр.: 11 назв.

Князев Н.А., Сальников А.Н. Управление динамическими приоритетами в планировании задач на многопроцессорных комплексах. //Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 64-73.

При работе нескольких пользователей с многопроцессорным комплексом, из-за ограниченности вычислительных ресурсов возникает проблема планирования запуска задач: чьи задачи запускать первыми, как оптимально построить расписание запуска задач, чтобы при этом учитывался их приоритет, и не происходило неоправданной долгой задержки больших задач в очереди.

В данной статье предлагается обзор современных подходов к планированию задач. В качестве решения этой проблемы предлагается алгоритм невозобновляемых фишек. В рамках этого алгоритма каждый пользователь, обладая ограниченным набором фишек, может добавлять их к своим запускаемым задачам. Задачи, имеющие большее число фишек, имеют больший приоритет. В качестве основного планировщика для реализации был выбран планировщик MAUI.

Ил.: 4 рис. Библиогр.: 11 назв.

Колесин М.С. Исследование возможностей применения алгоритма колонии пчел для решения задачи нахождения оптимального мэппинга параллельной задачи на архитектуру BlueGene/P. //Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 74-90.

В данной работе производится обзор эволюционных алгоритмов и применение алгоритма колонии пчел для решения задачи нахождения оптимального мэппинга параллельной задачи на архитектуру BlueGene/P. Приводятся результаты испытаний и дальнейшие возможные направления исследований.

Ил.: 9. Библиогр.: 9 назв.

Вылиток А.А., Сутырин П.Г., Харченков С.Л., Юдочев Д.В. О сопряжениях графовых описаний формальных языков. // Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 91-99.

В статье вводится понятие графового описания формальных языков, определяемое как обобщение понятий конечного автомата и D-графа. Рассматривается задача построения сопряжений графовых описаний, предлагается алгоритм построения компонентов сопряжений.

Библиогр.: 9 назв.

Арефьев Н.В., Мальковский М.Г. Синтаксический анализатор Treevial. Оценка семантической корректности синтаксической структуры. //Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 100-107.

В основе синтаксического анализатора Treevial, осуществляющего анализ текстов на естественном языке, лежит алгоритм перебора, управляемого эвристической функцией. В данной

статье обсуждается проблема оценки правильности синтаксических структур, возникающих в процессе перебора, и предлагается эвристика, позволяющая обнаружить большую часть неправильных (семантически некорректных) структур, а значит сократить время и улучшить качество анализа за счет отсеивания таких структур. Эта эвристика принимает во внимание три типа ограничений на сочетаемость слов в предложении (морфо-синтаксические, лексические и семантические) и оценивает, насколько построенные синтаксические структуры удовлетворяют этим ограничениям.

Библиогр.: 9 назв.

Павлов А.С. Исследование устойчивости метода обнаружения поискового спама на основе статистических характеристик. // Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 108-119.

В рамках данной работы излагается подход к обнаружению спам-текстов, обладающих локальной связностью, но нарушающих другие закономерности, характерные для естественных текстов. Предлагаемый подход основывается на использовании большого числа статистических характеристик текстов и машинного обучения. В статье описываются применяемые характеристики текстов и оценивается их значимость для задачи обнаружения поискового спама. Также экспериментально подтверждается устойчивость предложенного метода, в условиях, когда спамеры пытаются обойти данный метод.

Ил.: 2. Библиогр.: 16 назв.

Жарков А.В. Пивоварчук Д.Г. Параллельный алгоритм решения задачи управления развитием инфраструктуры типа «поставщик-потребитель». //Программные системы и инструменты. Тематический сборник № 10. М.: Изд-во факультета ВМиК МГУ, 2009. С. 120-124.

Работа посвящена исследованию параллельного алгоритма решения оптимизационной задачи на параллельных системах. Под инфраструктурой "поставщик-потребитель" в данной работе понимается сеть поставок, объединяющая некоторое количество поставщиков, потребителей и дистрибьюторов. Инфраструктура представляет собой набор заданных связей между поставщиками, дистрибьюторами и потребителями. Развитием инфраструктуры будем называть увеличение пропускных способностей связей в течение заданных моментов времени. Под оптимизацией развития сети поставок понимается её развитие с целью максимального удовлетворения спроса потребителей.

Ил.: рис: 3. Библиогр.: 7 назв.

Волканов Д.Ю., Зорин Д.А. Исследование применимости моделей оценки надёжности для разработки программного обеспечения с открытым исходным кодом. // Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 125-134.

В статье рассматриваются различные модели оценки надежности, с целью определения применимости моделей к проектам с открытым исходным кодом и уточнения механизма их использования описываются эксперименты с некоторыми крупными проектами с открытым исходным кодом. В результате экспериментов получены различные характеристики надежности рассматриваемых проектов и сформулированы гипотезы об общих свойствах проектов с открытым исходным кодом.

Ил.: 3 рис. Библиогр.: 12 назв.

Леонов М.В., Корнев Д.Б., Страхов В.Н. Пакет KSWeb: новый инструмент разработки мобильных информационных систем. // Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 135-139.

Сообщается об опыте разработки Web-серверного пакета KSWeb с Web-сервером, реализованном на языке Java и интегрированного с интерпретатором PHP и СУБД MySQL. Представленный пакет используется для разработки информационной системы для автоматизации историко-архивных исследований и для студенческого практикума по Web-технологиям.

Ил.: 1 рис. Библиогр.: 6 назв.

Шумкин Г.Н. Математическое моделирование молекулярных переключателей.// Программные системы и инструменты. Тематический сборник № 10, М.: Изд-во факультета ВМиК МГУ, 2009. С. 140-149.

Создание молекулярных устройств, которые можно переключать между состояниями с высоким и низким сопротивлениями, является одним из важнейших направлений нанотехнологий. Так как миниатюризация этих технологий продолжает развиваться, фундаментальная проблема идентифицирования и понимания наименьших физических систем, которые способны к переключательному поведению, привлекает растущий интерес. Последние эксперименты в лаборатории в научно-исследовательской лаборатории IBM в Цюрихе связаны с открытием индуцированной током таутомеризации водорода и наблюдением переключения проводимости молекул нафталоцианина.

Данная статья посвящена вычисления высоты барьера реакции таутомеризации в молекуле нафталоцианина с целью дальнейшего

изучения найденного переключателя и последующего практического применения.

Ил.: 3 рис. Библиогр.: 18 назв.

Научное издание

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 10

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

Подготовка оригинал-макета:
Глазкова Е.А.

Напечатано с готового оригинал-макета

Издательство ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 22.12.2009 г.
Формат 60x90 1/16. Усл.печ.л. 9,75. Тираж 100 экз. Заказ 736.
119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус, 627 к.
Тел. 939-3890, 939-3891. Тел /Факс 939-3891.