Безошибочное решение задач линейного программирования на многопроцессорных системах

В.В. Горбик

В работе исследуется проблема точного решения задач линейного программирования симплекс методом. Для осуществления вычислений без округлений применяются библиотеки точных вычислений *GNU MP* и *Rational*. Основная проблема производительности решается путем создания масштабируемого параллельного алгоритма симплекс метода, ориентированного на многопроцессорную среду с интерфейсом *MPI*. Новизна работы в синтезе точного и параллельного подходов к решению задач линейного программирования.

1. Введение

Объектом исследования данной статьи является параллельный алгоритм решения задач линейного программирования, точность и скорость вычислений. Используются классы безошибочных дробно-рациональных вычислений ***rational*** [1] и ***mpq_class*** из библиотеки ***GNU MP*** [2]. Важным аспектом при этом является возможность и эффективность адаптации данных классов к многопроцессорной среде. Интерфейс ***MPI*** уже на протяжении длительного времени является негласным стандартом при построении распределенных вычислительных систем. В работе рассмотрен способ интеграции классов точных вычислений к ***MPI*** на основе сериализации объектов.

2. Программные реализации

За годы разработок накопился большой опыт в решении задач линенейного програмирования. Есть алгоритмы, которые ориентиванны на разряженные матрицы, адаптированные к наименьшему накоплению погрешностей вследствие применения вычислений с плавающей точкой. Разнообразие вычисляющих кодов привело к появлению общего формата ***MPS*** [3], разработанного ***IBM*** еще в 60-х, который стал стандартом среди всех профессиональных продуктов и используется повсеместно до сих пор. Для тестирования оптимизационных программ существует библиотека ***Netlib*** [4], в которой хранятся в свободном доступе реальные задачи линейного программирования (в формате ***MPS***) разной размерности. Кроме самих задач присутствует описание и решение, полученное двумя признанными программными продуктами (***CPLEX*** [5] и ***MINOS*** [6]).

Список профессиональных коммерческих программных продуктов для решения задач линейного программирования довольно широк. Наиболее часто используемые можно найти в [7]. Представляет интерес тот факт, что все данные продукты, в том числе ***CPLEX*** и ***MINOS***, используют неточные вычисления с плавающей точкой. Например, в описании ***Netlib*** можно найти список задач, при решении которых продукты ***CPLEX*** и ***MINOS*** выдают результаты со значительным расхождением, либо вообще решение не может быть найдено.

Кроме фирменного программного обеспечения имеется ряд открытых кодов, решающих задачи линейного программирования [7]. Однако подавляющее большинство программных продуктов также производит вычисления с использованием типов данных с плавающей точкой, что приводит с неправильным решениям некоторых задач даже небольшой размерности, особого вида [8, 9]. Данная проблема касается всех программ решающих приближенно. Ведь в таком случае практически невозможно определить точность и правильность решения. Хотя, можно организовать дополнительную процедуру верификации решения по первоначальным условиям задачи. Погрешности скадываются из ограничений исходящих из самого стандарта ***IEEE*** [10] floating point, в частности из-за дробей, которые невозможно представить конечным числом знаков после десятичной запятой. Хотя относительная ошибка между представляемым числом и его реальным представлением не более чем $\xi = 2^{-52} = 1/4503599627370496 \approx 10^{-15,65}$, это

накладывает ограничения на точность вычислений в алгоритмах использующих типы данных с плавающей точкой.

Исключением из ряда программ использующих вычисления с плавающей точкой являют два открытых продукта ***EXLP*** [8] и ***QSopt-Ex*** [9], основанный на ***QSopt*** [11].

EXLP существует с 2002г. и до текущего момента приобрел мощный и оптимизированный код, умеет применять ряд правил для выбора ведущего столбца и сокращения числа итераций. Код написан на языке программирования ***C***, с использованием библиотеки точных вычислений ***GNU MP*** .

QSopt-Ex – это модифицированная версия ***QSopt***, где операции с плавающей точкой заменены использованием той же библиотеки точных вычислений ***GNU MP***. Данная программа является частью работы [12] и короткий вариант в виде исследования AT&T Labs [13].

2.1 Библиотеки точных вычислений

Типы данных, способные точно представлять дробно-рациональные числа можно реализовать с помощью двух векторов переменной длины (для числителя и знаменателя). Для целей безошибочных вычислений были выбраны библиотеки ***Rational*** и ***GMP*** в виду того, что ***Rational*** свободно доступен для внесения доработок**.** ***GMP*** также является открытой разработкой, входит в дистрибутивы ***GNU/Linux*** и имеет хорошую производительность.

Библиотека ***rational*** представляет из себя ***C++*** класс ***rational*** [1], построенный на объектах класса ***overlong***.

- Пакет ***gmp*** содержит ***GNU MP*** – открытую библиотеку для точных арифметических вычислений, операций над целыми числами со знаком, рациональными числами и числами с плавающей запятой [2]. С помощью класса ***mpq_class*** реализованы точные вычисления с дробями. Но в отличие от ***rational***, ***mpq_class*** построен на основе структур и функций ***C***-библиотеки ***gmp***, числителем и знаменателем являются структуры ***mpz_struct***, ***mpq_class*** представляет собой обертку для структур и функций работы с дробями ***gmp***. Идейно классы ***mpq_class*** и ***rational*** схожи, но реализованы по-разному.

3. Распараллеливание решения ЗЛП

Есть ряд программ решающих задачи линейного программирования и использующих преимущества распараллеливания [7]. Их немного и они не используют точные вычисления.

Для передачи пользовательских типов фиксированной длины в ***MPI*** предусмотрен специальный механизм описания. В случае с ***rational*** и ***mpq_class*** данный подход не пригоден, поэтому передача сводится к сериализация объектов в буфер с целью дальнейшей отправки (сериализация выгоднее чем несколько транзакций). Для этого в интерфейсы классов ***rational*** и ***mpq_class*** расширены методами, приведенными на рис. 1.

```
// serializeing and deserializeing for mpi
int object_size();
int serialize(void *buff, int sizeofbuff);
int deserialize(void *buff);
```

Рис. 1. Методы для адаптации к **MPI**

Дополнительные функции ***serialize_array()*** и ***deserialize_array()*** (см. рис. 2) осуществляют упаковку массивов объектов классов ***rational*** и ***mpq_class*** произвольной длины в единый буфер и распаковку из буфера.

```
char *serialize_array(mpt *r, int count, int &buffsize);
void deserialize_array(void *buff, mpt *r, int count);
```

Рис. 2. Дополнительные функции для массивов

Передачу буфера можно осуществлять стандартными средствами, но при этом возникают определенные трудности с определением необходимого размера буфера на стороне приемника. Данная проблема решается вызовом стандартной функции ***MPI_Probe***, которая позволяет уз-

нать размер пришедших данных. После этого можно выделить необходимый буфер, получить данные, и десериализовать объект или массив объектов. Для удобства реализованы так называемые «обертки» для функций передачи/приема ***MPI***. В результате такого подхода использование классов точных вычислений в среде ***MPI*** «прозрачно» для программиста и не требует внесения многочисленных изменений в код программы, например, при замене вещественных типов данных в существующем алгоритме на точные.

3.1 Параллельная версия алгоритма

В случае с табличным симплекс методом, в связи со спецификой вычислений, удачнее будет декомпозиция по столбцам. В данном случае все столбцы, коэффициенты целевой функции, делятся в равных пропорциях между процессами, вектор базисных переменных и правых частей рассылаются всем и обрабатываются отдельно. Матрица формируется изначально таким образом, чтобы каждому процессу досталось примерно равное количество столбцов исходной задачи и столбцов вводимых на этапе порождения начального базисного допустимого решения.

В случае с модифицированным симплекс методом, изначальная матрица должна быть известна всем процессам, т.к. заранее не известно какая переменная войдет в базис и каким процессом будет обрабатываться. Кроме того, итерационная процедура основана на том, что на каждом шаге известны: базисные переменные и их значения, обращение базиса, соответствующие базису симплекс множители. Дополнительные накладные расходы на коммуникацию при пересчетах приводят к тому, что не удается создать эффективную параллельную версию алгоритма [15].

Изложенный подход к распараллеливанию симплекс метода, был реализован в виде ***MPI*** программы ***plinpex*** *(parallel lineal exact solver)*, использующей для точных, дробно рациональных вычислений библиотеки ***Rational*** или ***GNU MP*** (задается опциями компиляции).

В качестве среды разработки использовался набор утилит ***GNU/gcc 4.1.1***, отладчик ***GNU/gdb***, профилер ***GNU/valgrind***. Для написания кода использовалась интегрированная среда разработки ***Eclipse 3.3*** (***CDT 4.0)***. Написание, тестирование и вычислительный эксперимент проводились на операционной системе ***Gentoo GNU/Linux***, на платформе ***i686***.

Рассмотрим ключевые моменты алгоритма работы ***plinpex***.

1. После запуска параллельных версий программы и инициализации среды ***MPI***, каждая из них идентифицирует себя по рангу.

2. Процесс с рангом 0 (часто его называет корневой), производит считывание входного файла задачи линейного программирования в формате ***MPS***. Остальные процессы переходят на шаг 3.

2.1. Последовательное считывание секций входного файла, запоминание имен переменных (они выводятся при распечатке решения), инициализация и заполнение матрицы коэффициентов и коэффициентов целевой функции, значений столбца свободных членов.

2.2. Расширение матрицы искусственными переменными для приведения задачи к нормальной форме и дополнительными переменными, необходимыми для порождения базисного плана. Инициализация вектора переменных, входящих в базис.

3. Происходит синхронизация процессов, после чего, одновременно вызывается основной метод ***Solve()***.

3.1. Широковещательная передача от корневого процесса общих параметров задачи, таких как:

- количество строк и столбцов;
- количество основных, искусственных и дополнительных переменных.

3.2. Каждый процесс, зная свой ранг и общее количество процессов, вычисляет принадлежащие ему основные столбцы и дополнительные (необходимого для порождения допустимого базисного плана).

3.3. Все процессы, кроме корневого, инициализируют ресурсы для приема матрицы коэффициентов, вектора свободных членов, искусственной и основной целевых функций, вектора переменных входящих в базис.

3.4. Широковещательная передача от корневого процесса вектора свободных членов и вектора переменных входящих в базис.

3.5. Поочередная рассылка корневым процессом основных и искусственных столбцов матрицы всем остальным процессам.

3.6. Синхронизация всех процессов. Перед основным циклом итеративной процедуры симплекс метода.

3.6.1. Каждый процесс выбирает один столбец из имеющихся у него столбцов на основе правила Данцига (минимальный отрицательный коэффициент целевой функции).

3.6.2. С помощью вызова ***MPI_Allreduce*** находится минимальный отрицательный коэффициент целевой функции среди всех процессов и ранг процесса, у которого находится данный столбец (назовем его ведущий процесс).

3.6.3. Если на шаге 3.6.2. среди всех столбцов не оказалось отрицательных коэффициентов целевой функции, то данный этап решения закончен, иначе переход на шаг 3.6.4.

3.6.3.1. Если закончен первый этап порождения допустимого базисного решения, то все процессы исключают из дальнейших вычислений искусственные столбцы, и заменяют искусственную целевую функцию на основную.

3.6.3.2. Если закончен второй этап, то решение закончено, переход на шаг 4.

3.6.4. Ведущий процесс выбирает переменную исключаемую из базиса.

3.6.5. Ведущий процесс широковещательно рассылает индексы переменных вошедших и вышедших из базиса, а также ведущий столбец.

3.6.6. Каждый процесс осуществляет построение новой канонической формы по правилам симплекс метода на имеющихся у него столбцах.

3.6.7. Каждый процесс проверяет, принадлежат ли ему переменные, исключаемые или вошедшие в базис, и изменяет при необходимости вектор базисных переменных.

3.6.8. Итерация закончена, переход на шаг 3.6.1.

4. Процесс с рангом 0 выводит результаты решения задачи.

5. Завершение функционирования среды ***MPI*** и завершение параллельных процессов.

4. Вычислительный эксперимент

Вычислительный эксперимент проводился на собранном автором кластере кафедры ЭММиС ЮУрГУ, состоящего из 10 узлов P4 2,40 ГГц / 512 Mb RAM, построенного на ***Gentoo GNU/Linux*** + ***MPICH*** [16].

$$ПРППК = 8 \cdot 2,4 \cdot 2 + 2 \cdot 2,8 \cdot 2 = 38,4 + 11,2 = 49,6 \text{ Gflops}$$

В качестве входных данных для вычислительного эксперимента использовались задачи линейного программирования из библиотеки ***Netlib***, данные задачи представлены в таблице 1.

Таблица 1

ЗЛП отобранные из библиотеки Netlib

Название задачи	Кол-во ограничений	Кол-во переменных	Кол-во ненулевых элементов	Оптимальное значение
SCSD6	148	1350	5666	5,0500000077E+01
SHARE1B	118	225	1182	-7,6589318579E+04
SCTAP1	301	480	2052	1,4122500000E+03

Данные задачи были выбраны по причине разного вида матриц. В ***SCSD6*** количество переменных существенно превышает количество ограничений, в то время как ***SCTAP1*** имеет форму близкую к квадратной. На начальном этапе тестирования, в связи с большим преимуществом в скорости счета с классами ***mpq_class*** по сравнению с ***rational*** (см. таблица 2), было принято решение проводить основные расчеты только с ***mpq_class***.

Таблица 2

Сравнение времени счета задач с классами rational и mpq_class

Название задачи	Кол-во ограничений	Кол-во переменных	Кол-во ненулевых элементов	Время счета с ***mpq_class***, с	Время счета с ***ratioanal***, с
AFIRO	28	32	88	0,05	1,14
ADLITTLE	59	97	465	6,15	81,62
BLEND	75	83	521	7,25	99,90

На рис. 3 – 8 приведены результаты экспериментов.

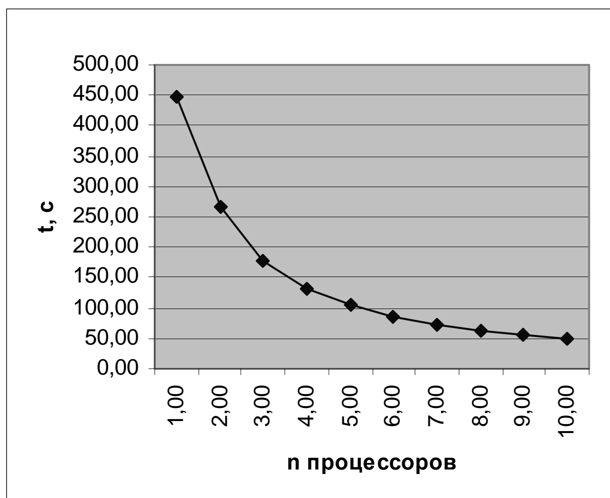

Рис. 3. Время счета на разном количестве процессоров для задачи SCSD6

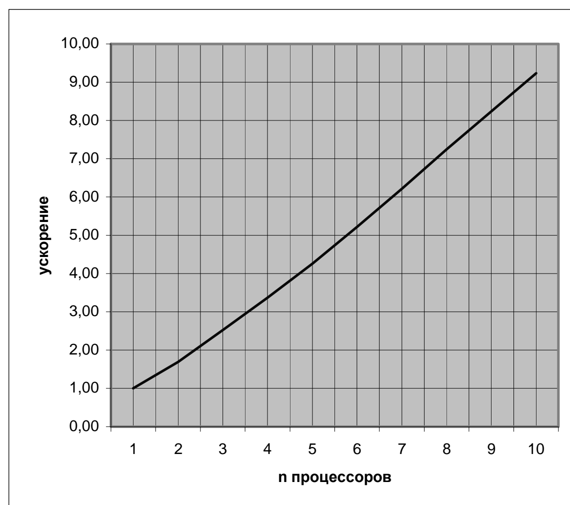

Рис. 4. График ускорения счета при введении дополнительных процессоров для задачи SCSD6

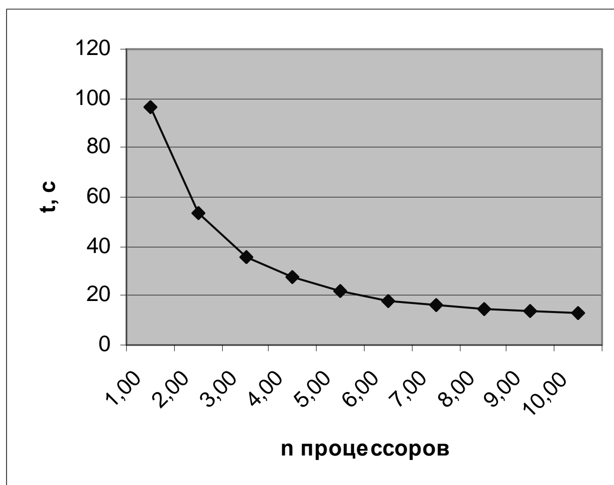

Рис. 5. Время счета на разном количестве процессоров для задачи SHARE1B

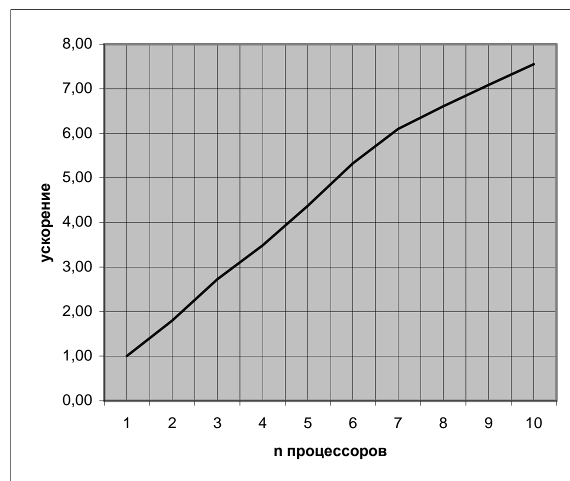

Рис. 6. График ускорения счета при введении дополнительных процессоров для задачи SHARE1B

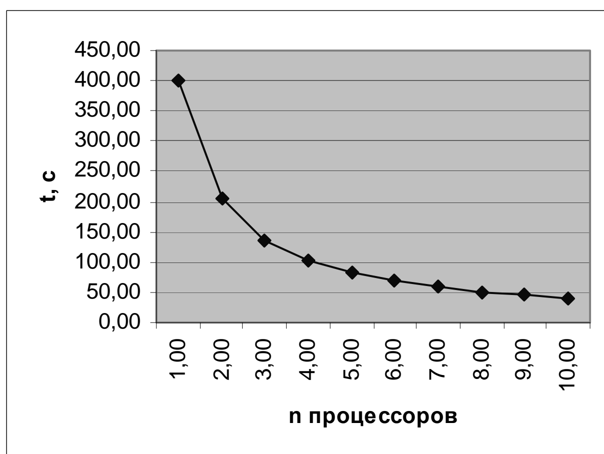

Рис. 7. Время счета на разном количестве процессоров для задачи SCTAP1

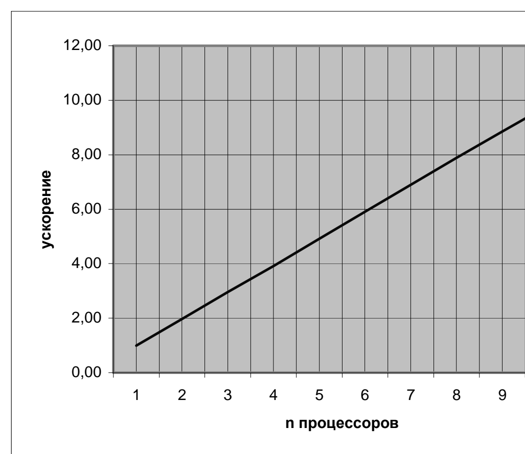

Рис. 8. График ускорения счета при введении дополнительных процессоров для задачи SCTAP1

Заключение

В результате проведенного эксперимента можно сделать выводы о высокой эффективности распараллеленного алгоритма симплекс метода. Использованный распараллеленный алгоритм использует всего две широковещательные коммуникации на каждой итерации основного цикла программы, что приводит к высокой оценке качества распараллеленного алгоритма. Проведенный эксперимент показал, что реализованный метод эффективен на задачах разной размерности и соотношением количества ограничений к количеству переменных. На сколько позволяют оценить имеющиеся вычислительные средства эффективность распараллеливания близка к линейной (на количестве узлов до 10).

Однако общее время выполнения алгоритма достаточно велико, и чтобы расширить диапазон решаемых задач, требуется дополнительная доработка в сторону оптимизации вычислений на узлах. Одним из направлений, на которое следует обратить внимание – критерий выбора ведущего столбца. В данном эксперименте использовалось классическое **правило Данцига**. Например, применяя правило **steepest edge** [15], время счета задачи ***SHARE1B*** на единственном процессоре сокращается с 96,84 до 26,03 секунд. Кроме того, принимая во внимание низкую производительность класса ***Rational*** (по сравнению с ***mpq_class***), можно отказаться от ***C***++ интерфейса библиотеки ***GNU MP*** и заменить математические операции с классами точных вычислений на предоставляющие больше гибкости и возможности оптимизации вызовы ***C*** процедур.

Литература

1. Панюков, А.В. Класс rational/ А.В. Панюков, М.И. Германенко, М.М. Силаев// Программы для ЭВМ. Базы данных. Топологии интегральных микросхем. – Официальный бюллетень Российского агентства по патентам и товарным знакам. №4(29), 1999 г. – М.ФИПС. – 1999. – Рег.№ 990607. – С. 97.
2. GNU Multiple Precision Arithmetic Library. – http://swox.com/gmp/
3. MPS format: – ftp://softlib.cs.rice.edu/pub/miplib/mps_format
4. Netlib collection. – ftp://netlib2.cs.utk.edu/lp/data
5. CPLEX. – http://www.ilog.com/products/cplex/
6. MINOS. – http://www.sbsi-sol-optimize.com/asp/sol_product_minos.htm
7. Fourer, R. Linear Programming Frequently Asked Questions/ R. Fourer// Optimization Technology Center of Northwestern University and Argonne National Laboratory, 2005. – http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html
8. EXLP. – http://members.jcom.home.ne.jp/masashi777/exlp.html
9. QSopt-Ex. – http://www.dii.uchile.cl/~daespino/
10. IEEE Computer Society, IEEE Standard for Binary Floating-Point Arithmetic, ieeestd 754-1985 ed., 1985.
11. QSopt. – http://www2.isye.gatech.edu/~wcook/qsopt/
12. Espinoza, Daniel G. On Linear Programming, Integer Programming and Cutting Planes/ Daniel G. Espinoza// School of Industrial and Systems Engineering Georgia Institute of Technology, 2006.
13. Applegate, David L. Exact solutions to linear programming problems/ David L. Applegate, William Cook, Sanjeeb Dash, Daniel G. Espinoza// AT&T Labs – Research, 2006.
14. Панюков, А.В. Класс overlong/ А.В. Панюков, М.М. Силаев// Программы для ЭВМ. Базы данных. Топологии интегральных микросхем. – Официальный бюллетень Российского агентства по патентам и товарным знакам. №4(29), 1999 г. – М.ФИПС. – 1999. – Рег. № 990486.
15. Yarmish, G. A Distributed Implementation of the Simplex Method/ G. Yarmish// UMI Dissertations Publishing, 2001.
16. MPICH. – http://www-unix.mcs.anl.gov/mpi/mpich1