Параллельный алгоритм нахождения общего решения системы линейных неравенств

Н.Ю. Золотых, С.С. Лялин

В статье рассматривается параллельный алгоритм одной модификации метода двойного описания многогранного конуса. Приводятся результаты вычислительного эксперимента на многопроцессорной машине с общей памятью и сравнение с другими реализациями алгоритмов, решающих данную задачу. Эксперименты показали как правило близкие к линейным результаты по масштабируемости.

1. Введение

Во многих прикладных задачах возникает такой объект как полиэдр – множество точек многомерного пространства, удовлетворяющих системе линейных неравенств. Например, в биологии статический анализ метаболических сетей приводит к решению системы линейных уравнений и неравенств. В программной оптимизации полиэдры возникают при рассмотрении линейного класса программ и его расширений.

Работа с полиэдрами, как правило, заключается в выполнении следующих операций: пересечение полиэдров, выпуклое объединение полиэдров, проекция на линейные многообразия, обход всех граней некоторой заданной размерности и т.д. В качестве одной из базовых задач, которую приходится решать при осуществлении перечисленных операций, выступает задача перехода между двумя представлениями полиэдра.

В данной статье даётся схематичное описание алгоритма, реализованного в программе Skeleton 3 – параллельной модификации алгоритма двойного описания многогранного конуса, который решает задачу перехода между двумя представлениями полиэдра. Сообщаются последние результаты по вычислительной эффективности в сравнении с программами других авторов, решающими ту же задачу, а так же масштабируемость Skeleton 3 при вычислении с вещественной арифметикой и арифметикой многократной точности.

2. Необходимые определения

Для краткости изложения будем рассматривать постановку задачи в однородной форме. Пусть дана матрица $A \in \mathbf{R}^{m \times d}$. Многогранным выпуклым конусом C в пространстве $\mathbf{R}^d$ называют множество решений системы линейных неравенств $Ax \geq 0$:

$$C = \left\{ x \in \mathbf{R}^d \mid Ax \geq 0 \right\}. \tag{1}$$

Кроме представления (1) конус C может быть записан в параметрическом виде, а именно, для некоторой матрицы $B \in \mathbf{R}^{d \times n}$:

$$C = \left\{ x \in \mathbf{R}^d \mid x = B\lambda,\ \lambda \in \mathbf{R}_+^n \right\}, \tag{2}$$

где $\mathbf{R}_+$ – неотрицательные действительные числа. Алгоритм двойного описания [12], известный также как алгоритм Моцкина-Бургера [6], по одному из описаний, (1) или (2), строит другое. При этом метод находит неприводимую систему векторов в матрице B (неприводимую систему неравенств в матрице A соответственно). Везде далее мы будем считать, что рассматривается задача перехода от (1) к (2). Для простоты изложения будем считать, что ранг матрицы A равен d. В этом случае неприводимая система векторов в описании (2) называется остовом конуса. Векторы остова (лучи) определяются единственным образом с точностью до умножения на положительные константы. Так как порядок строк в A и столбцов в B,

очевидно, не имеет значения, то далее для упрощения изложения символы A и B используются также для обозначения множества строк и столбцов соответствующих матриц.

3. Существующие алгоритмы и реализации

Авторам известны несколько других алгоритмов (и соответствующих реализаций), решающих поставленную задачу. Родственным алгоритмом, который построен на тех же идеях, что и описываемый алгоритм, является еще одна модификация [10] алгоритма [12], реализация которой, cdd и cdd+, доступна в Интернет [9]. Хорошо известны ещё два алгоритма: lrs [8] и qhull [13]. Все перечисленные реализации доступны в последовательной форме. В [1] описана параллельная реализация алгоритма [12].

4. Алгоритм

Рассматриваемый алгоритм является параллельной модификацией алгоритма [12] построенной на основе модификации [2] и реализации [14]. Предварительные результаты по улучшению версии [2] уже докладывались ранее в [3, 4, 5].

Опишем схему алгоритма. Оригинальный алгоритм [2] имеет итеративный характер: на каждой итерации рассматривается очередное неравенство из матрицы A и проводятся все необходимые вычисления, после чего алгоритм переходит к следующему неравенству. При переходе с итерации на итерацию перестраиваются две структуры данных: текущий остов конуса B (множество лучей) и текущий подграф графа смежности лучей E (множество пар лучей). Перечислим основные стадии, из которых состоит одна итерация алгоритма:

Алгоритм 1. Последовательный алгоритм построения остова многогранного конуса.

0. *Начальный симплициальный конус*. Выбрать произвольно невырожденную квадратную подматрицу A_0 размера d матрицы A. Положить $B = A_0^{-1}$ и E равным всевозможным упорядоченным парам из B.
1. *Выбор неравенства*. Выбирается очередное неравенство из A. Хотя неравенства могут выбираться не последовательно, в данной работе рассматривается только последовательный выбор (в том порядке, в котором идут строки в A).
2. *Классификация*. Существующие лучи конуса B разделяются на три группы: (а) удовлетворяющие выбранному неравенству строго, (б) удовлетворяющие неравенству как равенству, (в) не удовлетворяющие неравенству.
3. *Получение новых лучей.* Перебирая все элементы из E, комбинируются существующие лучи, входящие в пары, и определённым образом порождаются новые лучи. Они включаются в B и добавляются в группу (б), созданную на стадии 2.
4. *Образование пар.* Подготавливаются пары из лучей группы (б), которые будут проверены на потенциальную смежность в следующей стадии.
5. *Потенциальная смежность.* Лучи попавшие в группу (б) попарно проверяются на потенциальную смежность в конусе.
6. *Смежность*. Для каждого луча группы (б) из числа потенциально смежных лучей определённым образом выбираются смежные лучи; все такие пары лучей добавляются в E.
7. *Удаление ненужных лучей.* Все лучи из группы (в) удаляются.

При завершении алгоритма, когда все неравенства A рассмотрены, B содержит остов конуса, т.е. описание (2).

На рис. 1. дана схема зависимостей между стадиями одной итерации алгоритма. Нисходящие зависимости (т.е. идущие от стадий с меньшими номерами к стадиям с большими номерами) указывают на зависимости стадий внутри одной итерации. Восходящие зависимости

(т.е. идущие от стадий с большими номерами к стадиям с меньшими номерами) указывают на зависимость следующей итерации от текущей итерации. Зависимость по данным, которая обозначена сплошной стрелкой на рис. 1. можно рассматривать как очередь из данных от одной стадии к другой стадии.

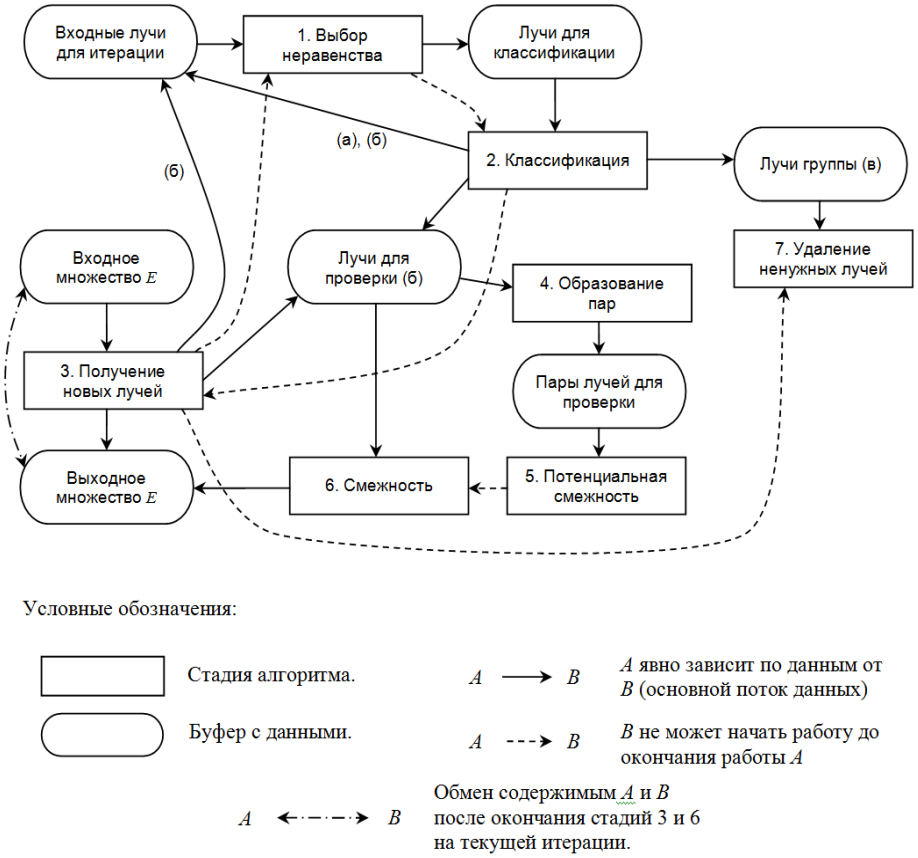

Рис. 1. Зависимости между стадиями алгоритма.

Анализ показал, что при наличии такого рода зависимостей можно так реализовать алгоритм, чтобы избежать обязательного упорядочивания работы по основным стадиям. Другим словами, часть работы стадии с большим номером может быть выполнена до окончания работы стадии с меньшим номером, если нет дополнительных зависимостей.

5. Параллельный алгоритм и его реализация

Работа каждой из стадии 1—7 в отдельности может быть организована параллельно. При этом если не пользоваться свойством указанным выше, в конце каждой стадии требуется точка синхронизации всех исполнительных устройств (барьер). Это ухудшает балансировку загрузки между вычислительными ресурсами: время работы одной стадии трудно предсказать и трудно (если вообще возможно) разделить данные между вычислительными устройствами так, чтобы добиться идеальной балансировки. Неизбежна ситуация когда в конце работы стадии многие вычислительные устройства будут простаивать (Рис. 3).

Использование свойства отсутствия обязательного упорядочивания между стадиями позволяет построить более эффективный параллельный алгоритм, так как устраняет обязательные барьерные синхронизации в конце каждой стадии (Рис. 4).

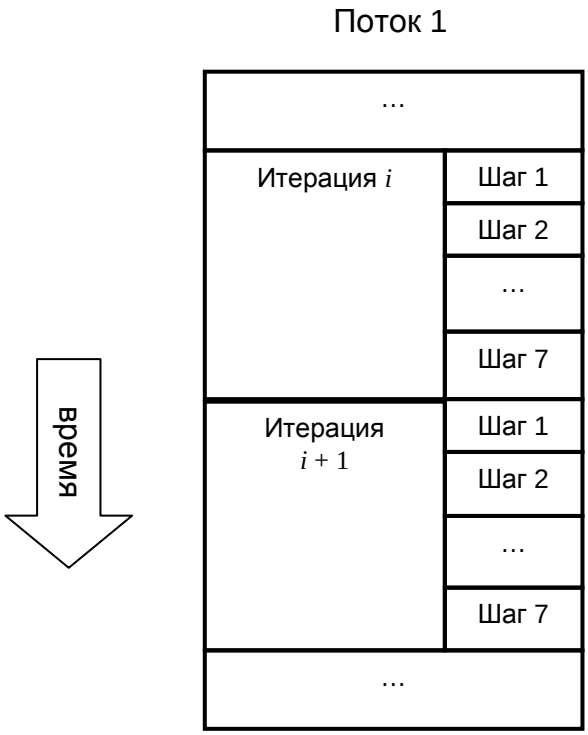

Рис. 2. Последовательный поток управления. Указаны номера шагов Алгоритма 1.

-

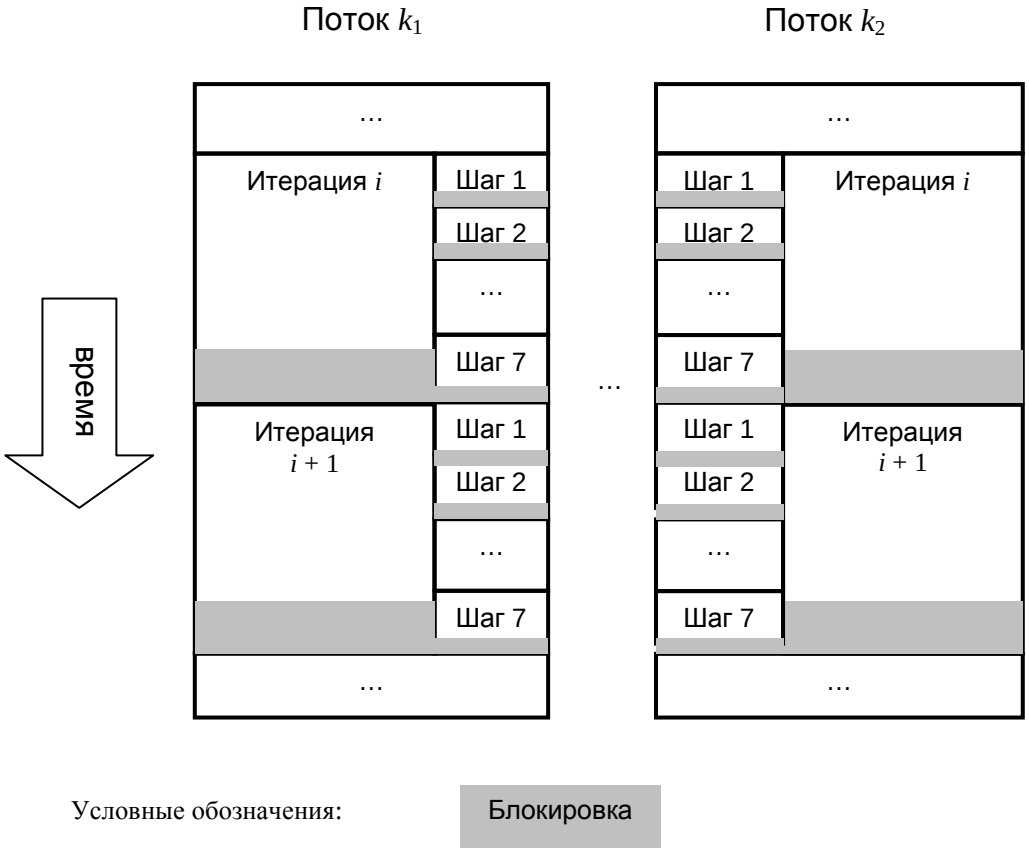

Рис. 3. Пример реализации параллельного потока управления при синхронизации в конце каждой стадии. Указаны номера шагов Алгоритма 1.

Skeleton 3 [15] – это реализация описанного алгоритма с использованием указанного свойства. Программа работает на многопроцессорной машине с общей памятью и использует дешёвый способ синхронизации через атомарно модифицируемые целочисленные переменные (interlocked-операции). Каждый из буферов (овалов на рис. 1) реализован в виде односвязного списка с неблокируемым доступом. Все потоки равнозначны и синхронизируются через небольшой набор глобальных переменных. Синхронизации типа «барьер» отсутствуют. Такое понятие как «текущая итерация» (итерация алгоритма, обрабатываемая в данный момент времени) для *всех потоков исполнения* теряет свой смысл, так как в текущий момент времени разные потоки могут обрабатывать данные с разных итераций. Понятие «текущая итерация для алгоритма» заменяется «текущей итерации для потока». Далее для краткости мы опускаем слово «текущий» в словосочетаний «текущая итерация» и «текущая стадия» имея ввиду ту итерацию и ту стадию, которые исполняет данный поток.

Итак, в описываемой реализации все потоки равнозначны и каждый исполняет представленный алгоритм следующим образом:

Алгоритм 2. Исполнение алгоритма 1 каждым потоком параллельной реализации.

Шаг 1. [*Выбор стадии*] Опираясь на метки-номера итераций для буферов данных, выбрать стадию алгоритма 1, которую можно исполнить и для которой сейчас есть данные. Если выбор сделан, то перейти на Шаг 2, если готовых к исполнению данных нет, то перейти к Шагу 4.

Шаг 2. [*Обработка*] Обработать одну порцию данных на выбранной стадии, после чего проверить условия завершения стадии для итерации.

Шаг 3. [*Завершение стадии*] Если условия завершения стадии на текущей итерации проверенные на Шаге 2 удовлетворяются, то поток завершает стадию локально (в контексте этого потока) и, если он последний завершает данную стадию (т.е. все остальные потоки уже завершили стадию), то стадия помечается завершённой для итерации глобально (для всех потоков). Если это не завершающая стадия на последней итерации, то переход на Шаг 1, иначе – завершение работы.

Шаг 4. [*Подглядывание*] Проверить, есть ли необрабатываемые порции данных у других потоков в каких либо стадиях, которые сейчас можно обработать. Если есть, то перейти на Шаг 5. Если нет – на Шаг 6.

Шаг 5. [*Заимствование*] Переместить порцию данных из другого потока в соответствующий буфер текущего потока и перейти к Шагу 2.

Шаг 6. [*Ожидание*] Сообщить операционной системе об отказе от текущего выделенного кванта времени (выполнить yield) ИЛИ Средствами операционной системы *пассивно* подождать период времени равный среднему времени обработки порции данных (требует сбор статистики по времени исполнения). Затем перейти на Шаг 1.

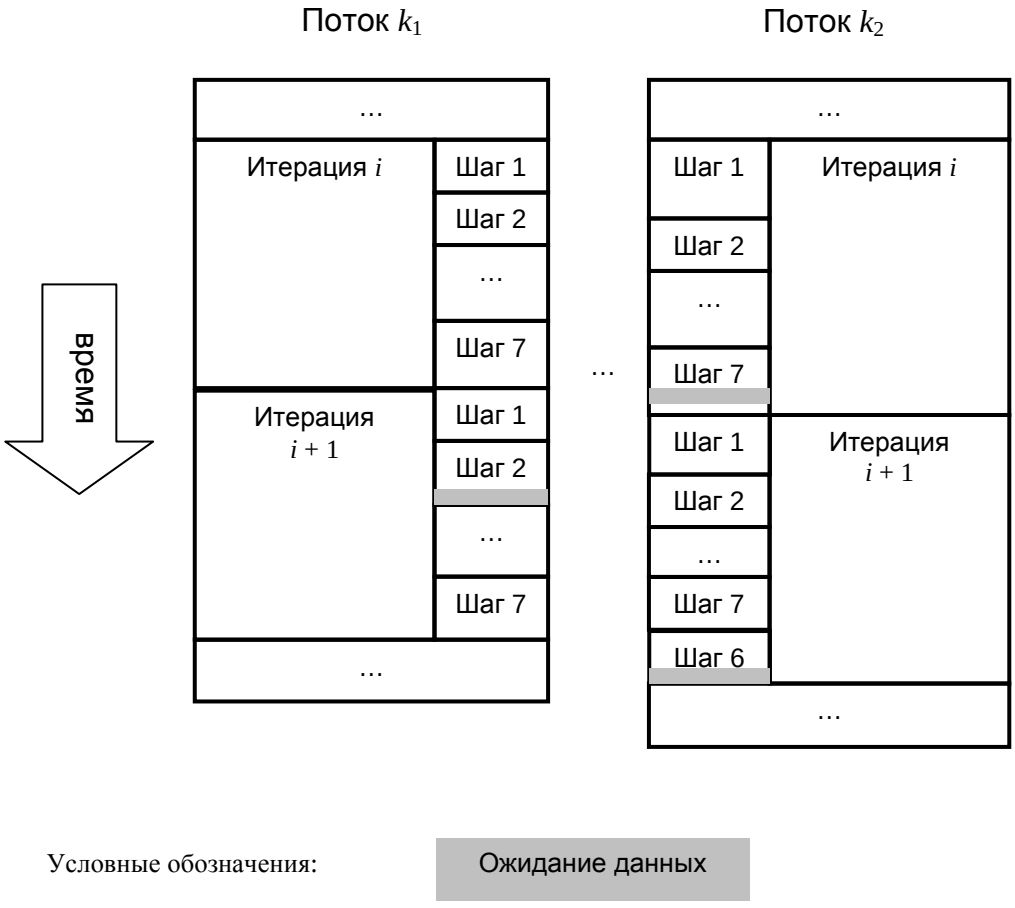

Рис. 4. Пример реализации параллельного потока управления алгоритма 2. Указаны номера шагов Алгоритма 1.

Как уже было описано в последней публикации [5], эффективность реализации по сравнению с первыми результатами Skeleton 3 [3] улучшилась. Более эффективное использование кэша процессора, некоторые низкоуровневые оптимизации и использование инструкций Intel® SSE и Intel® SSE 2 позволило ускорить работу Skeleton 3 по сравнению со своим предшественником Skeleton 2 [2, 14] в несколько раз при исполнении на одном потоке (см. таблицу 1). При этом нельзя сказать об однозначно хорошей масштабируемости

программы, так как есть такие наборы данных, на которых Skeleton 3 показывает плохие результаты по этому параметру (например, cube16.ine; см. рис. 5 и рис. 6).

Для сравнения производительности и оценки масштабируемости использовались несколько наборов данных, которые позаимствованы из дистрибутива cdd [9] и фигурируют в различных сравнительных исследованиях программ построения остова многогранного конуса; см. таблицу 1. В экспериментах использовались две различные реализации арифметики: 1) вещественная арифметика двойной точности выраженная типом double в C++ и 2) целочисленная «длинная» арифметика GMP [11] подключённая к библиотеке Arageli [7].

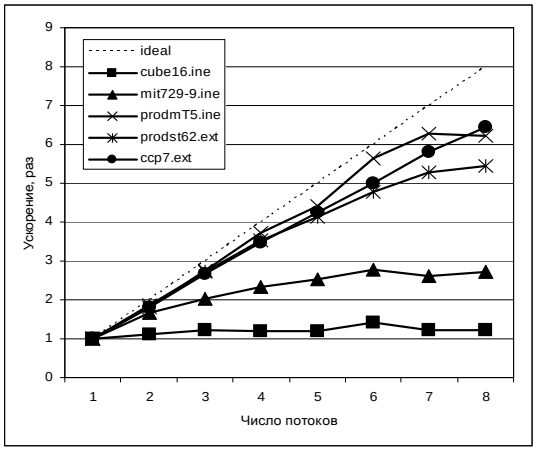

Рис. 5. Масштабируемость Skeleton 3 на вещественной арифметике.

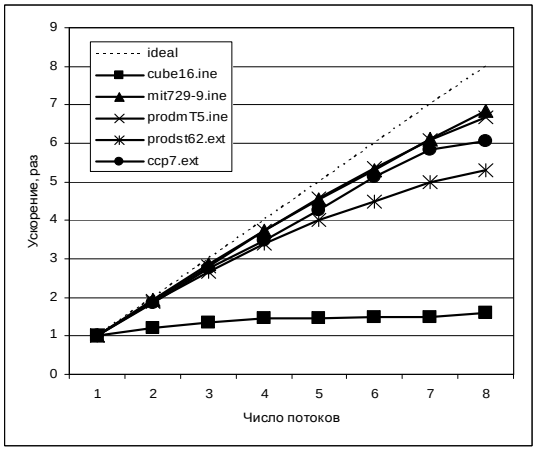

Рис. 6. Масштабируемость Skeleton 3 на целочисленной арифметике.

Таблица 1. Время работы Skeleton 2 и Skeleton 3 на нескольких наборах данных.

Набор данных	Параметры задачи			Время работы Skeleton 2 (вещественная арифметика), сек	Время работы Skeleton 3 на одном потоке (вещественная арифметика), сек	Время работы Skeleton 3 на одном потоке (целочисленная арифметика GMP), сек
	d	m	n			
cube16.ine	17	32	65536	5.959	4.548	6.929
mit729-9.ine	9	729	4862	54.147	15.368	731.327
prodmT5.ine	20	711	76	845.146	269.169	863.840
prodst62.ext	25	3461	168	1195.584	320.192	2352.040
ccp7.ext	22	64	116764	10257.774	2269.150	2539.730

Вычисления производились на машине с двумя четырёхядерными процессорами Intel® Xeon® X5355 2.66 ГГц, 8 ГБайт оперативной памяти. Реализация использует потоки Win32 API операционной системы Microsoft Windows и библиотеку Arageli [7]. На рис. 5 и рис. 6 показаны графики масштабируемости Skeleton 3 при запуске на различном числе ядер для вещественной (рис. 5) и целочисленной арифметики (рис. 6).

При сравнении с программой cdd [9] было получено, что на одних наборах данных быстрее работает Skeleton 3 (например, cube16.ine и mit729-9.ine), а на других – cdd (например, prodst62.ext). Это связано с одним нюансом в работе рассматриваемого алгоритма и cdd, который связан с порядком проверки критериев отбора подходящих пар лучей на стадиях 4 и 5 основного алгоритма 1. В настоящий момент авторам известна модификация, которая улучшит работу Skeleton 3 на наборах подобных prodst62.ext, что позволит приблизиться или обогнать cdd на таких примерах.

Skeleton 3 доступен on-line через сервис, разработанный С.В. Лобановым [16].

6. Реализация алгоритма для распределённой памяти

Продолжается работа над параллельной версией алгоритма для машин с распределённой памятью. Используется та же схема, что приведена на рис. 1, но появляются дополнительные стадии по асинхронной пересылке данных с узла на узел. Проводятся эксперименты по спекулятивной (упреждающей) посылке данных, чтобы минимизировать потерю производительности из-за большого времени ожидания коммуникационной сети.

С этой же целью проверяется возможность спекулятивных вычислений, когда один узел, обрабатывая какую-то порцию данных, ещё не знает, что какой-то другой узел начал обрабатывать ту же порцию данных. Эта ситуация возникает, как следствие спекулятивной посылки данных, а так же в случае применения разных эвристик на шагах 1, 4, 5 и 6: при избытке вычислительных ресурсов на недостаточно больших задачах можно использовать разные группы вычислительных узлов для различных эвристик. Какая группа вычислит результат быстрее, тот результат и берётся для продолжения вычислений, а на всех остальных группах вычисления отменяются. Данный подход позволяет значительно увеличить скорость работы для задач, для которых не известно поведение Skeleton, следовательно, неизвестно, какая из эвристик даст результат за минимальное время.

Литература

1. Агафонов Е.А., Земскова Е.Л., Золотых Н.Ю. Параллельный алгоритм построения остова многогранного конуса и его программная реализация. Информационный бюллетень Ассоциации математического программирования № 10. Конференция "Математическое программирование и приложения" (тезисы докладов). – Екатеринбург: УрО РАН, 2003. – С.15–16.
2. Золотых Н.Ю. Новая модификация метода двойного описания для построения остова многогранного конуса. III Всероссийская конференция «Проблемы оптимизации и экономические приложения». Тезисы докладов. – Омск, 2006. – С. 108.
3. Золотых Н.Ю., Лялин С.С. Оптимизация одной модификации алгоритма двойного описания для многогранного конуса. Технологии Microsoft в теории и практике программирования. Материалы конференции. Под ред. проф. Р.Г. Стронгина. – Нижний Новгород: изд-во Нижегородского госуниверситета, 2007. – С. 333–336.
4. Лялин С.С. Параллельная модификация алгоритма двойного описания для многогранного конуса. Высокопроизводительные параллельные вычисления на кластерных системах. Материалы Седьмой Международной конференции-семинара. Под ред. проф. Р.Г. Стронгина. – Нижний Новгород: Изд-во Нижегородского госуниверситета, 2007. – С. 240–243.
5. Лялин С.С. Skeleton 3 – реализация параллельного алгоритма построения остова многогранного конуса. Высокопроизводительные параллельные вычисления на кластерных системах. Материалы Восьмой Международной конференции-семинара.– Казань: Изд-во КГТУ, 2008. – С. 167–171.
6. Черников С.Н. Линейные неравенства. Москва: Наука, 1968.
7. Arageli Home Page: http://www.arageli.org.
8. Avis D. lrs: A Revised Implementation of the Reverse Search Vertex Enumeration Algorithm. In: Polytopes - Combinatorics and Computation, G. Kalai & G. Ziegler eds., Birkhauser-Verlag, DMV Seminar Band 29, P. 177-198, 2000.
9. Cdd Home Page: http://www.ifor.math.ethz.ch/~fukuda/cdd_home/cdd.html.
10. Fukuda K., Prodon A. Double description method revisited. In Deza M., Euler R., and Manoussakis I., editors, Combinatorics and Computer Science, volume 1120 of Lecture Notes in Computer Science, P. 91–111. Springer-Verlag, 1996.
11. GNU Multiple Precision Arithmetic Library Home Page: http://gmplib.org.
12. Motzkin T.S., Raiffa H., Thompson G.L., Thrall R.M.. The double description method. In H. W. Kuhn and A. W. Tucker, editors, Contributions to the Theory of Games – Volume II, number 28

in Annals of Mathematics Studies, P. 51–73. Princeton University Press, Princeton, New Jersey, 1953.
13. Qhull Home Page: http://www.qhull.org.
14. Skeleton 2 Home Page: http://www.uic.nnov.ru/~zny/skeleton.
15. Skeleton 3 Home Page: http://www.arageli.org/skeleton.
16. Skeleton 3 on-line demonstration: http://www.arageli.org/skeletondemo.