Развитие параллельных методов вычисления базисов Грёбнера и инволютивных базисов*

Д.А. Янович
Лаборатория информационных технологий ОИЯИ, Дубна

Модулярное вычисление базисов [1]

Введение

Ввиду широко известной проблемы разрастания промежуточных коэффициентов при построении базисов Грёбнера и инволютивных базисов, в мире имеется большой интерес к применению модулярных методов, позволяющих ее избежать.

Огромным преимуществом применения модулярных вычислений является естественный параллелизм, за счет слабой связанности подзадач друг с другом обещающий хорошую масштабируемость.

Для построения базиса (Грёбнера или инволютивного) с помощью модулярных вычислений необходимо выполнить следующие шаги [2, 3]:

1. Вычислить достаточное количество модулярных образов базиса идеала, порожденного исходной системы.
2. Выполнить восстановление коэффициентов из кольца $\mathbb{Z}$ у всех термов.
3. Проверить, является ли полученное множество полиномов базисом вообще и базисом идеала, порожденного исходной системой, в частности. Если не базис, повторяем с пункта 1.

Как видно, все шаги легко допускают параллелизацию: на первом и третьем шаге очевидную, второй шаг можно распараллелить по схеме, похожей на умножение Карацубы.

Кроме плюсов, модулярный подход имеет по крайней мере два минуса, отсутствующие при вычислениях с коэффициентами из $\mathbb{Z}$.

Выполнение первого шага сразу же ставит перед нами проблему «неудачных» модулей – вычисление образов по некоторым модулям приводит к потере информации. В данном контексте это означает, что некоторые термы, имеющие ненулевые коэффициенты в результирующем базисе, отсутствуют в модулярном образе и не могут быть восстановлены.

*Исследования, представленные в работе, были выполнены при частичной поддержке грантов 10-01-00200 и 12-07-00294 Российского Фонда Фундаментальных Исследований и гранта 3802.2012.2 Министерства Образования и Науки Российской Федерации.

Второй проблемой является адекватное восстановление полиномов из их образов. При «наивном» подходе можно подумать, что проводя вычисления над кольцом $\mathbb{Z}_p[x]$ мы сможем по китайской теореме об остатках получить коэффициенты исходного полинома. Однако, каждый образ f_p полинома f, вычисленный по разным модулям p, будет иметь в своем составе коэффициент $c_p \cdot f_p$, зависящий от модуля. Этот коэффициент неустраним (у нас нет модулярного аналога наибольшего общего делителя), поэтому надо переходить к вычислениям надо кольцом рациональных чисел, при которых коэффициент лидирующего терма всегда будет равен единице.

В данной работе коэффициенты в образах принадлежат множествам $\mathbb{F}_p$, числам Фарея, получение образа и восстановление рационального числа описано в [4].

Вопросы эффективной реализации

Во первых, каждое вычисление образа базиса по модулю вообще может дать разные результаты (с точки зрения мономов, находящихся в полиномах) при использовании разных модулей. Однако, чтобы получить не вероятный, а действительный базис, мы обязаны восстановить и проверить все полученные образы.

Для быстрого разбиения образов на классы автор использовал сравнение полинома Гильберта (для определения эквивалентности лидирующих мономов) и *подпись базиса* для проверки идентичности мономов в хвостах.

Определение 1 *Пусть дан базис* $F = f_1, \ldots, f_n$, *полиномы которого сортированы по возрастанию лидирующих мономов по какому-либо допустимому упорядочению. Назовем подписью базиса значение контрольной суммы* `md5` *[5], полученное суммированием всех мономов всех полиномов* $f_i | \forall i = 1, \ldots, n$, *причем мономы суммируются в порядке убывания по тому же упорядочению.*

Коллизии, существующие в `md5`, хотя и редкие, но вероятные, обходятся сравнением полиномов Гильберта. Таким образом, используя подпись и полином Гильберта можно эффективно и точно разделить образы базисов по мономам, т.е.

отсечь «неудачные» модули, поскольку соответствующих им образов будет очень мало.

Во вторых, оказалось важно, сколько модулярных образов базисов надо вычислить перед началом восстановления. Пример *chemequs* в однопоточном варианте с подсчетом 8 базисов за проход, считается около получаса на Xeon-E5410@2.33Ггц. Если же считать по 80 базисов за проход, время вычислений на той же машине снижается до одной минуты.

В третьих, необходимо уделять внимание порядку применения китайской теоремы об остатках для восстановления коэффициентов из $\mathbb{Z}$. Необходимо, чтобы модулярные коэффициенты обрабатывались строго в порядке возрастания модулей. Несоблюдение этого условия вызывает существенное падение производительности на всех рассмотренных примерах.

Алгоритм легко можно представить по следующему наброску:

1. Определяем количество базисов n, считаемых перед этапом восстановления
2. Ставим в очередь задачи вычисления отдельных базисов
3. Классифицируем полученные базисы используя подпись и полином Гильберта
4. Если ни в одном классе не достигнута мощность n переходим на пункт 2
5. Выполняем восстановление всех классов базисов с мощностью n, сначала по китайской теореме об остатках, потом восстанавливаем $\mathbb{Z}$-коэффициенты из чисел Фарея
6. Проверяем, что восстановленный базис действительно является базисом, используя модулярную проверку [3]. Если не является базисом - пункт 1
7. Проверяем, что он является базисом идеала, порожденного исходной системой полиномов. Вычисления выполняем в кольце $\mathbb{Z}$. Если не является базисом - пункт 1
8. Проверяем, что восстановленный базис действительно является базисом, используя вычисления над $\mathbb{Z}$. Если не является базисом - пункт 1

Результаты

Описанный алгоритм был реализован на языке C в программе `ModJB`, используя библиотеку `pthreads` для машин с общей памятью [1]. На сервере 2xXeon-E5410@2.33Ггц под управлением Gentoo Linux были вычислены базисы для тестовых примеров из баз [6, 7]. Для сравнения была взята реализация алгоритма, входящая в систему компьютерной алгебры `Singular 3.1.3` [3], использована старшая стабильная версия на момент проведения исследования. В таблице участвуют примеры со временем вычисления базиса Грёбнера однопоточной версией системы `Singular` меньшим двух часов и большим 15 секунд.

Как видно из таблицы 1, время вычисления однопоточной версии программы автора проигрывает системе `Singular`, но уже на восьми ядрах виден существенный выигрыш у этой системы. Общий коэффициент ускорения `ModJB` на 8 ядрах составляет около 7 раз, а реализация `Singular` не дотягивает и до полуторакратного ускорения. Также интересно, что на некоторых примерах (в таблице они выделены жирным шрифтом), автором получено суперлинейное ускорение. Такое уже встречалось как в собственной прошлой работе [8], так и в работах других авторов – на небольшом примере в [9], причина этого явления требует отдельного исследования.

Исследование масштабируемости вариантов параллельного алгоритма на массивном многоядерном SMP компьютере [10]

В ходе вычислений базисов Грёбнера и инволютивных базисов почти все время последовательный алгоритм проводит в вычислениях отдельных инволютивных нормальных форм. Распределив их вычисление по разным ядрам, получим первый алгоритм для исследования, детально описаный в работе [11].

В качестве соперника выберем описаный выше вариант параллельного алгоритма, использующий вычисления с модулярными коэффициентами.

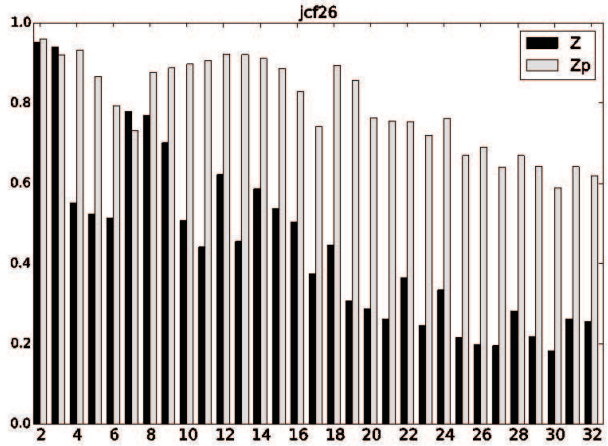
Реализации этих алгоритмов были протестированы на 32-х ядерных серверах, предоставляемых сервисом *Elastic Cloud Computing*, использовался шаблон `cc2.8xlarge`, соответствующий 32х-ядерному серверу с процессорами Xeon E5-2670@2.60GHz с 64Гб RAM. Коэффициенты масштабирования двух характерных примеров (с большим временем счета в последовательном варианте) отображены на следующих диаграммах.

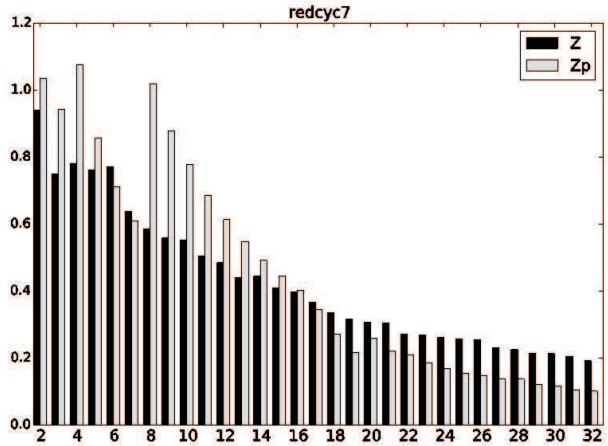
`Jcf26` представляет класс примеров с большой промежуточной арифметикой. Как видно, примеры такого класса демонстрируют отличную масштабируемость в случае использования модулярного алгоритма и постоянное, но не сильное, снижение эффективности на процессор в случае вычислений в "полной"арифметике.

`Redcyc7` является представителем примеров с малой промежуточной арифметикой. Как видно из диаграммы, примеры данного типа оправдано

Пример	1 поток		8 потоков		Ускорение	
	ModJB	Singular	ModJB	Singular	ModJB	Singular
assur44	24.99	21.72	3.84	13.09	6.51	1.66
chemequs	460.09	324.06	66.92	384.00	6.88	0.84
chemkin	78.63	327.46	11.39	383.95	6.90	0.85
cohn3	388.66	123.92	57.17	105.41	6.80	1.18
cyclic7	230.80	74.16	48.10	40.77	4.80	1.82
d1	163.07	50.03	24.95	47.21	6.54	1.06
dl	461.34	50.47	78.84	41.36	5.85	1.22
eco10	152.79	71.38	20.96	49.49	7.29	1.44
eco11	3240.40	1467.70	391.06	787.56	8.29	1.86
extcyc6	991.09	333.78	227.68	185.67	4.35	1.80
f855	336.12	44.64	49.39	31.89	6.81	1.40
fabrice24	207.19	134.25	32.36	120.25	6.40	1.12
hcyclic7	280.83	1118.52	52.78	1047.14	5.32	1.07
i1	2447.78	1528.42	325.26	1514.99	7.53	1.01
ilias13	4473.00	640.51	790.14	422.58	5.66	1.52
ilias_k_2	807.22	167.87	128.73	117.05	6.27	1.43
ilias_k_3	1284.32	285.68	222.26	207.64	5.78	1.38
jcf26	1380.28	447.62	99.01	430.81	13.94	1.04
katsura8	287.41	62.22	27.99	40.35	10.27	1.54
katsura9	3240.73	707.83	487.53	409.26	6.65	1.73
kin1	1364.09	957.05	185.26	951.50	7.36	1.01
noon8	2570.20	145.30	405.64	150.92	6.37	0.96
rbpl24	220.86	133.86	32.26	120.46	6.87	1.11
rbpl	2277.37	768.07	283.91	725.19	8.02	1.06
redcyc7	119.96	50.20	13.20	94.15	9.09	0.53
redeco11	754.67	142.92	81.18	108.89	9.30	1.31
reimer6	64.34	218.64	9.96	115.54	6.46	1.89
virasoro	41.61	15.31	5.80	13.21	7.17	1.16
Всего	28349.84	10413.59	4163.57	8660.33	6.81	1.20

Таблица 1: Сравнение коэффициентов ускорения программы автора и пакета `Singular`





считать на небольших машинах с 8-12 ядрами, не более.

Как показывают диаграммы, масштабируемость тем лучше, чем длиннее промежуточные коэффициенты примера. В этом случае наиболее оправдано применять модулярный алгоритм, который позволяет избежать их вычисления вовсе. Если промежуточные коэффициенты неве-

лики, можно применять любой вариант алгоритма, они обеспечат достаточно заметный прирост скорости. Также видно, что практически любой пример получит хорошее ускорение на легко доступных 8-ми ядерных машинах.

Список литературы

[1] *Янович Д.А.* Параллельное модулярное вычисление базисов Грёбнера и инволютивных базисов // Программирование. 2013, №2, с.75–80

[2] *Arnold E.A.* Modular algorithms for computing Gröbner bases // Journal of Symbolic Computation 35, 403–419 (2003).

[3] *Idrees N., Pfister G., Steidel S.* Parallelization of Modular Algorithms. // (arXiv:1005.5663), Journal of Symbolic Computation 46, 672-684 (2011).

[4] *Kornerup P., Gregory R.* Mapping integers and Hensel codes onto Farey fractions // Bit 23, 9—20 (1983).

[5] `http://en.wikipedia.org/wiki/MD5`

[6] *Bini D., Mourrain B.* Polynomial Test Suite. `http://www-sop.inria.fr/saga/POL`

[7] *Verschelde J.* The Database with Test Examples. `http://www.math.uic.edu/~jan/demo.html`

[8] *Гердт В.П., Янович Д.А.* Параллельное вычисление базисов Жане и Гребнера над полем рациональных чисел // Программирование. 2005, №2, с.73–80

[9] *Leykin A.* On parallel computation of Gröbner bases // In Proceedings of ICPP 2004 workshops, High Performance Scientific and Engineering Computing, IEEE Computer Society, 2004, p.160—164.

[10] *Янович Д.А.*Исследование масштабируемости параллельных вычислений инволютивных базисов и базисов Грёбнера на многоядерном SMP компьютере // Сборник трудов конференции ММСР-2013.

[11] D. A. Yanovich: Reduction-Level Parallel Computations of Gröbner and Janet Bases. *Bulletin of Peoples' Friendship University of Russia*, Mathematics. Information Sciences. Physics. No.3, Issue 2 (2010), pp. 19–24.