Масштабируемость статических алгоритмов использующих метод гомоморфных образов *

Г. И. Малашонок , С. А. Хворов

malaschonok@gmail.com, griffter@yandex.ru

Аннотация. Посвящается исследованиям в области параллельной компьютерной алгебры, в частности статическому распараллеливанию алгоритма обращения целочисленной матрицы.

Классический подход к решению полиномиальных и целочисленных задач компьютерной алгебры состоит в применении метода гомоморфных образов. Он заключается в выборе нужного числа конечных полей, решению задачи в каждом конечном поле и восстановлению результата в требуемой области. Такой подход позволяет не только уменьшить общую сложность вычислений, но и проявить естественный параллелизм, который позволяет применять схемы статического распараллеливания. Мы рассматриваем случай, когда задача в каждом конечном поле может быть решена на одном процессоре кластера. Такой же подход можно применить и для матриц большего размера. Для обращения матрицы в кончном поле потребуется выделить не отельные ядра, а изолированные коммуникаторы состоящие из групп ядер одинакового размера. В каждом таком коммуникаторе можно вычислять обратную матрицу с использованием динамического или статического распараллеливания. Все остальные вычисления со статической схемой управления сохраняются.

Мы приводим результаты экспериментов с матрицами в пределах 10^3 на кластере, в котором менялось число ядер от четырех до 10^3. и демонстрируем масштабируемлсть алгоритма в широком диапазоне.

Полученные результаты позволяют предсказывать время работы и рекомендуемое число ядер кластера для обращения матриц больших размеров. Следующие серии экспериментов можно проводить на кластере, у которого число ядер порядка $10^4 - 10^5$.

Ключевые слова

параллельная компьютерная алгебра, распределенная память, обращение матрицы, статическое распараллеливание, конечные поля, метод гомоморфных лбразов

1 Введение

Задача обращения матрицы является одной из тех задач, которые используются в очень многих приложениях. Для вычисления обратных матриц небольших размеров достаточно воспользоваться любым алгоритмом типа Гаусса или его моификациями, а для хранения данных использовать стандартные форматы для чисел с плавающей точкой.

Для матриц большого размера можно воспользоваться итерационными алгоритмами, но только в том случае, если на входе задана подходящая матрица и алгоритм сходится. Но что делать в общем случае?

Первое, что можно сделать – это попытаться найти результат любым известным методом. Мы ожидаем, что будет выполняться равенство $A^{-1}A = I$. Если матрица $Er = A^{-1}A - I$ удовлетворительно близкая к нулевой, то задача решена. Но к сожаленью, с ростом числа (n) строк и столбцов, матрица ошибок Er все сильнее отличается от нулевой. И попытки использовать привычные алгортмы со сложностью $\sim n^3$ обречены на неудачу.

Можно было бы искать обратную матрицу, используя характеристический полином. Известно, что характеристический полином матрицы A обращается в ноль при подстановке в него матрица A. Если

* Данная работа была частично поддержана грантом РФФИ 16-07-00420

умножить на A^{-1} уравнение $charPol(A) = 0|_{x=A}$, то получается выражение для обратной матрицы, в котором она получается линейной комбинацией $n-1$ матриц, каждая из которых является степенью матрицы A. Если ограничиться только вычислением всех степеней матрицы A от 2 до $n-1$, то это уже n^4 операций над элементами. Если принять во внимание сложность быстрого матричного умножения (n^β), то получим всего $n^{\beta+1}$ операций над коэффициентами для вычисления всех степеней матрицы A. Это при том, что проблема накопления погрешности и проблема поиска достаточно точного выражения для характеристического полинома сохраняется.

На этом фоне привлекательно выглядит алгоритм, в котором всего $\sim n^4m^2$ бит-операций (m – число двоичных разрядов у коэффициентов матрицы), который дает точный ответ и обеспечивает естественное статическое распараллеливание.

Мы приводим такой алгоритм и результаты кластерных экспериментов.

2 Алгоритм

Алгоритм основан на методе гомоморфных образов, который часто используют при решении задач компьютерной алгеры [1]-[3]. Он заключается в том, задача решается в отдельных конечных полях, а затем по найденным решениям в конечных полях находится ответ в требуемой области. При этом не только уменьшается общая сложность вычислений, но еще проявляется и естественный параллелизм.

Мы рассматриваем случай, когда задача в каждом конечном поле может быть решена с применением одного ядра. Переход к задачам большего размера может быть осуществлен путем создания в кластере изолированных коммуникаторов. Размеры коммуникаторов должны выбираться так, чтобы число ядер в нем было достаточным для решение задачи в простом поле (с помощью динамического или статического распараллеливания). Во всем остальном данный алгоритм не потребует изменений.

Мы опираемся на рекурсивный алгоритм [4], который вычисляет определитель $\det A$ матрицы A и присоединенную матрицу A^* и имеет сложность матричного умножения. Обратная матрица вычисляется как $A^{-1} = A^*/\det A$.

Мы полагаем, что входная матрица целочисленная. Переход к конечным полям сводится к вычислению остатков при делении. Когда результат найден в каждом конечном поле, то остается восстановить каждый элемент матрицы, как целое число, по его известным остаткам.

Пусть $m_1, m_2, ..., m_n$ – различные простые числа, превосходяшие 2, $M = m_1m_2...m_n$ – их произведение, $r_1, r_2, ..., r_n$ – целые числа и пусть n_{ij} – это обратный к m_i элемент в простом конечном поле из m_j элементов. Как известно, существует единственное целое число в интервале $[-(M-1)/2, (M-1)/2]$, которое по модулю m_i имеет остаток r_i, $(i = 1, 2, ..n)$. Его можно найти по такому алгоритму, который обычно называют алгоритмом Ньютона:
$c_1 = r_1; \quad \overline{c}_1 = c_1 \ mod \ m_2,$
$c_2 = c_1 + (r_2 - \overline{c}_1)m_1n_{12}; \quad \overline{c}_2 = c_2 \ mod \ m_3 \quad$ и так далее,
$c_n = c_{n-1} + (r_n - \overline{c}_{n-1})m_1...m_{n-1}n_{1,n}...n_{n-1,n},$

Результат: $x = c_n \ mod \ M$, а общее число бит-операций $\sim (\log_2 M)^2$.

Для оценки наибольшего числа, которое может появиться в результате, используется неравенство Адамара, которое оценивает определитель матрицы $A = (a_{i,j})$ порялка n: $|det(A)| \le \prod_{i=1}^{n}(\sum_{j=1}^{n} a_{i,j}^2)^{1/2} \le n^{n/2} \max(|a_{i,j}|)^n$.

Следовательно, для восстановления одного числа потребуется не более $\sim n^2(log_2(\sqrt{n}) + m)^2$ бит-операций, где $m = \log_2 max|a_{i,j}|$ – разрядность наибольшего коэффициента матрицы A.

Выбор множества простых чисел осуществлялся так. Пусть дан набор всех простых чисел $p_1 < p_2 < \ldots < p_{m-1} < p_m < 2^{32}$. Будем выбирать из него простые числа до тех пор, пока для их произведения не станет выполняться неравенство: $\prod_{i=q}^{m} p_i < 2|det(A)|$. Дополним полученный список простых чисел так, чтобы общее количество было кратно числу ядер кластера. Каждому ядру достанется одинаковое число простых чисел из этого списка.

В таком алгоритме общее количество бит-операций будет $\sim (mn^{\beta+1} + n^4m^2)$. Здесь n – размер матрицы, а m – разрядность коэффициентов входной матрицы $A = (a_{i,j})$: $m = max_{i,j}(log_2|a_{i,j}|)$, если оценка сложности производится в числе бит-операций и величиной $\log_2 \sqrt{n}$ мы пренебрегли.

В экспериментах мы использовали стандартное матричное умножение и числа для исходной матрицы A брали с $m = 28$. Поэтому ожидаемое число бит-операций для этих двух этапов, соответственно, равно $\sim n^4 m$ и $\sim n^4 m^2$, а общая оценка сложности такого алгоритма в бит-операциях будет $\sim n^4 m^2$.

Алгоритм состоит из следующих 5 этапов:

1. Нулевой процессор рассылает исходную матрицу A всем остальным процессорам. Это одна коллективная операция обмена.
2. Каждый процессор $\mathbf{Pr}_k$ выбирает свой интервал простых чисел и для каждого из них вычисляет присоединенную матрицу и определитель в простом поле.
3. Процессоры обмениваются строками полученных матриц так, чтобы у каждого было примерно одинаровое число строк или одинаковое число элементов (в случае разреженных матриц). Это одна коллективная операция обмена (AllToAllv()).
4. Каждый процессор восстанавливает свой фрагмент обратной матрицы.
5. Все процессоры отсылают нулевому процессору свой фрагмент обратной матрицы. (Если результат может привысить размеры оперативной памяти на нулевом процессоре, то это выполняется в асинхронном режиме, чтобы сохранять фрагменты на жестком диске.)

Такой алгоритм для обращения матрицы обсуждался раннее в работах [5] - [7].

3 Результаты экспериментов

Все эксперименты проводились на кластере МВС-10П в МСЦ РАН.

Этот кластер имеет 207 вычислительных узлов. Каждый из узлов оснащен 2 процессорами по 8 ядер на каждом. Общее количество оперативной памяти на одном узле равно 64 гигабайтам.

Чтобы можно было анализировать эффективность программы на каждом из пяти этапов, производилось измерение времени, которое затрачено на каждом этапе.

В первой серии экспериментов плотная матрица размера 1024 обращалась на 4 разных по числу ядер вычислителях. Результаты представлены в таблице 1. В этой таблице показано, сколько времени занимают отдельные участки параллельной программы. Приведены результаты вычислений с использование на 128, 256, 512 и 1024 ядер.

Номер эксперимента	1	2	3	4
Число ядер в эксперименте	**128**	**256**	**512**	**1024**
1. Рассылка матрицы с 0 на все процессоры	5	5	6	6
2. Вычисление в конечных полях	291	$160_{(97\%)}$	$89_{(81\%)}$	$47_{(77\%)}$
3. Рассылка строк между процессорами	4	6	7	8
4. Восстановление всех элементов	74	$38_{(91\%)}$	$20_{(93\%)}$	$11_{(84\%)}$
5. Сборка результата на 0 процессоре	14	13	11	11
Итого: общее время вычисления	388	$222_{(83\%)}$	$133_{(73\%)}$	$83_{(58\%)}$

Таблица 1. Время (сек.) обращения плотной целочисленной матрицы размера 1024×1024. Показано общее время и отдельное время для каждого этапа алгоритма. В скобках указана эффективность распараллеливания по отношению к 128 ядрам: $k = \frac{t_{128}}{t_N} \frac{128}{N} \cdot 100\%$, отдельно для этапов 2, 4 и общая.) .

В скобках указана эффективность по отношению к случаю с 128 ядрами. Хорошо видно, что второй и четвертый этапы имеют высокую эффективность распараллеливания, но итоговая эффективность несколько хуже из-за затрат на пересылку данных между ядрами.

В таблице 2 приведены результаты экспериментов с разными размерами матриц и разным числом ядер. Исходные матрицы, как и в первой серии экспериментов, были плотные и имели целые коэффициенты с 28-разрядными двоичными числами.. Для срвнения приведено время, которое требуется для чисто последовательной программы, которая была запущена на 1 ядре с 8 гигабайтами ОЗУ. Во всех экспериментах на каждое ядро кластера приходилось по 8 гигабайт ОЗУ.

n \ p	S	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
8	0.1	0.2	0.1						
16	0.2	0.3	0.2	0.1					
32	0.2	0.4	0.3	0.3	0.2				
64	1.2	0.9	0.8	0.7	0.5	0.2			
128	30	4.5	3	1.8	1.5	1.2	0.8		
256	557	50	28	14	8	5.5	4	2.5	
512	4621	756	404	202	102	50	28	17	9

Таблица 2. Время обращения плотной целочисленной матрицы. Обозначено: p (по горизонтали) - число процессоров, n (по вертикали) - размер матрицы, S - время вычисления для последовательного алгоритма. Время измеряется в секундах.

n \ p	**8**	**16**	**32**	**64**	**128**	**256**	**512**
32	(**67**%)	33%	25%				
64	(**56**%)	32%	23%	28%			
128	75%	(**63**%)	38%	23%	18%		
256	89%	89%	78%	(**57**%)	39%	31%	
512	94%	94%	93%	95%	84%	69%	(**66**%)

Таблица 3. По таблице 2 расчитана эффективность распараллеливания по отношению к 4 ядрам: $k = \frac{t_4}{t_N}\frac{4}{N} \cdot 100\%$. Круглыми скобками выделено рекомендуемое пользователю число ядер для заданного размера матрицы (это максимальное число ядер при эффективности более 50%.

По Таблице 2 можно сравнить время которое необходимо для обращения матриц размера 256 и 512 в экспериментах с одинаковым числом ядер. Так как размеры матриц отличаются в 2 раза, то число операций должно отличаться примерно в $2^4 = 16$ раз. Проследим во сколько раз отличается время вычисления в экспериментах с 4, 8, .. 256 ядрами:

$$[\frac{756}{50}, \frac{404}{28}, \frac{202}{14}, \frac{102}{8}, \frac{50}{5.5}, \frac{28}{4}, \frac{17}{2.5}] = [15.1,\ 14.4,\ 14.4,\ 12.8,\ 9.1,\ 7.0,\ 6.8].$$

Как видно из этого сравнения, в экспериментах с 4, 8, 16 и 32 ядрами это отношение находится в пределах интервала $[15.1, 12.8]$, что близко к теоретическому значению 16. Для большего числа ядер, как видно из таблицы 1, время, которое необходимо для пересылок данных становится все ближе к чистому времени вычислений. Добавочная константа становится велика, поэтому это отношение постепенно уменьшается.

Еще две серии экспериментов проводились для разреженных матриц, у которых плотность, то есть отношение числа ненулевых элементов к числу всех элементов, составляла 50% и 20%.

n \ p	S	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
8	0.1	0.2	0.1						
16	0.2	0.3	0.2	0.1					
32	0.2	0.4	0.3	0.3	0.2				
64	1.1	0.8	0.7	0.6	0.5	0.2			
128	26	4	2.5	1.5	1.3	1	0.7		
256	524	41	23	12	7	5	3	2	
512	4357	627	337	166	84	42	23	13	7

Таблица 4. Время обращения целочисленной матрицы с плотностью 50%: p - число процессоров, n - размер матрицы, S - время вычисления для последовательного алгоритма.

n \ p	**8**	**16**	**32**	**64**	**128**	**256**	**512**
32	(**67**%)	33%	25%				
64	(**57**%)	33%	20%	25%			
128	80%	(**67**%)	38%	25%	18%		
256	89%	85%	73%	(**51**%)	43%	32%	
512	93%	94%	93%	93%	85%	75%	(**70**%)

Таблица 5. По таблице 4 расчитана эффективность распараллеливания по отношению к 4 ядрам для матриц с плотностью 50%. Круглыми скобками выделено рекомендуемое пользователю число ядер для заданного размера матрицы (это максимальное число ядер при эффективности более 50%.

n/p	S	**4**	**8**	**16**	**32**	**64**	**128**	**256**	**512**
8	0.1	0.2	0.1						
16	0.1	0.2	0.2	0.1					
32	0.2	0.3	0.2	0.2	0.1				
64	1	0.6	0.5	0.4	0.3	0.1			
128	22	3	2	1.3	1	0.7	0.4		
256	504	32	19	8	5	4	2.5	1.5	
512	4033	502	267	135	68	32	17	10	5

Таблица 6. Время обращения целочисленной матрицы с плотностью 20%: p - число процессоров, n - размер матрицы, S - время вычисления для последовательного алгоритма.

n \ p	**8**	**16**	**32**	**64**	**128**	**256**	**512**
32	(**75**%)	37%	37%				
64	(**60**%)	38%	25%	37%			
128	76%	(**69**%)	38%	27%	23%		
256	84%	100%	80%	(**50**%)	40%	33%	
512	94%	93%	92%	98%	92%	78%	(**78**%)

Таблица 7. По таблице 6 расчитана эффективность распараллеливания по отношению к 4 ядрам для матриц с плотностью 20%. Круглыми скобками выделено рекомендуемое пользователю число ядер для заданного размера матрицы (это максимальное число ядер при эффективности более 50%.

4 Заключение

Отметим, что таблицы 3,5 и 7 очень близки. У них совпадает рекомендуемое число ядер для всех размеров матриц. Для матриц размера 32 и 64 можно брать 8 ядер. Для матриц размера 128 – 16 ядер. Для матриц размера 256 – 64 ядера и для матриц размера 512 можно брать 512 ядер. При таком выборе числа ядер эффективность параллельных вычислений не будет ниже 50%.

В целом алгоритм ведет себя достаточно предсказуемо. В первой таблице приводится время, которое необходимо для вычислений на каждом из 5 отдельных этапов алгоритма. Как видно по этой таблице, на втором и на четвертом этапах обеспечивается эффективное масштабирование. А эффективность масштабирования всего алгоритма убывает по мере того, как время счета становится соизмеримым со временем пересылки данных.

Этот факт указывает только на то, что размер кластера должен выбираться в зависимости от размера входной матрицы. Применение большого числа ядер для матриц малых размеров не зффективно.

Результаты экспериментов показывают, что вполне можно ставить задачу обращения матриц значительно больших размеров и прогнозировать необходимое число ядер и время вычислений. Но для этого

нужно иметь возможность проводить эксперименты на более мощном кластере. Следующим этапом мог бы быть кластер с числом ядер порядка: $10^4 - 10^5$.

Отметим также в заключение и вторую важную особенность алгоритма – это отсутствие погрешности при вычислениях. Вычисляемая обратная матрица имеет дробные коэффициеты. При умножении на исходную матрицу получается единичная матрица без погрешности. Если пользователю нужно менее точное значение коэффициентов, то можно осуществить округление коэффициентов. При необходимости, можно исследовать как округление будет влиять на точность решения прикладной задачи.

5 Благодарности

Авторы выражают благодарность руководству МСЦ РАН за предоставленную возможность проводить вычисления на кластере МВС-10П.

Список литературы

[1] Компьютерная алгебра. Символьные и алгебраические вычисления. Под.ред.Б.Бухбергера, Дж.Коллинза, Р.Лооса. М.: МИР, 1986.

[2] Малашонок Г.И. Матричные методы вычислений в коммутативных кольцах. Монография. Тамбов: Изд-во ТГУ им. Г.Р.Державина, 2002.

[3] Малашонок Г.И. Дискретная математика с элементами компьютерной алгебры: Учебное пособие. Тамбов : Изд-во ТГУ им. Г.Р.Державина, 2005.

[4] Малашонок Г.И. О вычислении ядра оператора, действующего в модуле. Вестник Тамбовского университета. Сер. Естественные и технические науки. Том 13, вып. 1, 2008. С.129-131.

[5] Malaschonok G.I. Khvorov S. Parallel Inversion of Integer Matrix. // International Conference on Mathematical Partnership, Parallel Computing and Computer Algebra: MathParCA-2015, Turku-Abo, Finland, 2015. P. 7.

[6] Хворов С.А. Параллельный алгоритм обращения целочисленной матрицы: эксперименты на кластере МВС-10П // International Conference on Mathematical Partnership, Parallel Computing and Computer Algebra: MathParCA-2017. P. 43-48.

[7] Хворов С.А. Параллельный алгоритм обращения целочисленной матрицы: результаты экспериментов // Вестник Тамбовского университета. Серия: Естественные и технические науки. Тамбов, 2018. Т. 23. № 121. С. 100-108.

[8] Open MPI: Open Source High Performance Computing [Электронный ресурс] // URL: http://www.open-mpi.org/software/ompi/v1.7/downloads/openmpi-1.7.4.tar.bz2

[9] MPI: The Message Passing Interface // URL: http://www2.sscc.ru/Links/Litera/tech/mpi.html [Электронный ресурс]

[10] Malaschonok G.I., MathPartner Computer Algebra, *Programming and Computer Software*, **43**, No. 2, 112–118, 2017.

[11] Малашонок Г.И. Руководство по языку “Mathpar“. Тамбов: Изд-во Тамбовского университета, 2013.

[12] Малашонок Г.И., Ильченко Е.А., Рыбаков М.А., Хворов С.А. Параллельное программирование на OpenMPI Java с приложением в Math Partner:. Учеб. пособие в 2 ч. Часть 2 Тамбов: Изд. дом ТГУ им. Г.Р. Державина, 2016

[13] Система компьютерной алгебры "MATH PARTNER"[Электронный ресурс]// URL://http://mathpar.cloud.unihub.ru

[14] Malaschonok G.I. Mathpar Language Guide. Tambov: Publishing House of TSU, 2013, 125 p.

[15] Малашонок Г.И., Переславцева О.Н., Ивашов Д.С. Параллельные символьные вычисления // Суперкомпьютерные технологии в науке, образовании и промышленности / Под редакцией: академика В.А. Садовничего, академика Г.И. Савина, чл.-корр. РАН Вл.В. Воеводина, -М.: Издательство Московского университета, 2013, 208 с.