Высопроизводительные алгоритмы обращения матриц на GPU*

Н.С. Недожогин, А.С. Сармакеева, С.П. Копысов

Институт механики УрО РАН

Обращение матрицы является важным этапом при численном решении задач, таких как решение систем линейных уравнений и построение предобуславливателей, вычисление дополнения Шура в методах декомпозиции области, цифровая обработка изображений и т. д. Разработка высопроизводительных параллельных алгоритмов обращения матриц связана с эффективным хранением и отображением алгоритмов на современные многоядерные архитектуры. Наряду с традиционными методами обращения — LU-факторизацией и методом Гаусса-Жордана, рассмотрены параллельные алгоритмы метода сопряженных градиентов и пополнения, в которых используются матрично-векторные и скалярные произведения эффективно выполняемые на многоядерных процессорах. В работе проведено сравнение на тестовых матрицах рассматриваемых методов на CPU и GPU.

1. Введение

При решении практических задач возникает необходимость выбора того или иного метода или его параллельной реализации. Появление программного обеспечения для вычислений общего назначения на графических устройствах (GPGPU) позволило на ряде задач, в том числе вычислительной линейной алгебры, получать ускорение вычислений в десятки и сотни раз, по сравнению с центральным процессором (CPU). Тысячи потоков GPU могут эффективно выполнять одновременно большое число простых арифметических операций, что характерно для мультипликативных и аддитивных операций с векторами и матрицами. Вместе с тем, последовательные операции и ветвления, характерные для разложения матриц на треугольные множители, выполняются GPU медленнее, чем ядрами CPU.

Данная работа ориентирована на эффективные реализации методов обращения симметричных разрежённых и положительно определенных матриц с использованием гибридных вычислительных узлов с GPGPU. В работе проведено сравнение алгоритмов обращения матриц, имеющих разреженную структуру. При реализации всех методов учитывалась структура хранения в сжатом формате.

2. Алгоритмы обращения матриц

Рассмотрим высокопроизводительные алгоритмы явного и неявного обращения матриц на гибридной архитектуре с выделением операций эффективно выполняемых на универсальных и графических процессорах. Хотя явное обращение матрицы во многих случаях может быть нецелесообразным, оно по-прежнему представляет интерес в различных прикладных областях, где встречаются большие разреженные и, порой, плохообусловленные матрицы.

LU-факторизация (LU). Для заданной невырожденной матрицы A существует разложение на верхнюю треугольную U и на нижнюю треугольную L матрицы. Если матрицы A, U, L обратимы, то справедливо следующее утверждение: $A^{-1} = U^{-1}L^{-1}$. Так как факторизация проводилась для матриц хранящихся в формате CSR, при построении L и U суммировались лишь те произведения, в которых участвовали ненулевые элементы исходной матрицы [1]. Рассматривалась только последовательная реализация LU-факторизации

*Работа выполнена при финансовой поддержке РФФИ (грант 14-01-31066-мол_а,14-01-00055-а, 13-01-00101-а) и программы Президиума РАН № 18 при поддержке УрО РАН (проект 12-П-1-1005).

на CPU, показывающая наилучшие результаты для малых и средних матриц. Особенности данного алгоритма не позволяют эффективно использовать все возможности массивно-параллельной архитектуры GPU.

Метод Гаусса-Жордана (GJ). Потенциально, метод Гаусса-Жордана больше подходит для вычислений на графических ускорителях, чем алгоритм основанный на LU-разложении. В рассматриваемом варианте все операции выполняются на центральном процессоре CPU. Так как преобладающие вычисления в данном методе содержат матричные операции, то в дальнейшем можно ожидать высокую производительность варианта для гибридной архитектуры при совместном использовании CPU+GPU. Однако, в работе [2] показано, что максимально получаемое трехкратное ускорение метода обращения Гаусса-Жордана достигается лишь при вычислениях с одинарной точностью и разделением операций выполняемых на CPU и GPU.

Метод сопряженных градиентов (CG). Рассматриваемый в работе алгоритм вычисления обратной матрицы состоит из решений матричной системы вида $AX = E$, где E — единичная матрица. Система эффективно решается на GPU предобусловленным алгоритмом сопряженных градиентов. Использование в тестах простейшего диагонального предобуславливателя связано с обеспечением минимальных вычислительных затрат (количества требуемой памяти, времени построения предобуславливателя и времени его решения) на GPU для матриц общего вида.

Реализация алгоритма более подробно рассмотрена в работе [3]. Отметим только, что оценка числа ненулевых элементов обратной матрицы затруднена и обращенную матрицу приходится хранить полностью, что накладывает существенные ограничения на используемую память при работе с GPU. В случае обращения матрицы методом сопряженных градиентов результат вычислений получается по столбцам, что позволяет сразу преобразовывать матрицу в сжатый формат и, тем самым, снижать ограничение на размер задачи, решаемой на GPU. В методе сопряженных градиентов операции скалярного произведения, нормы векторов, суммы векторов и умножения вектора на скаляр распараллелены с помощью библиотеки cuBLAS. Матрично-векторное произведение реализовано с помощью технологии CUDA в векторном варианте с возможностью использования нескольких GPU.

Метод пополнения (SM). Идея метода пополнения [4] восходит к работе [5]. За основу построения A^{-1} берут матрицу B той же размерности, что и A, но с известной обратной матрицей.

Теорема [5]. Пусть B — невырожденная матрица и вектора $\mathbf{u}$ и $\mathbf{v}$ такие, что

$$r = 1 + \mathbf{v}^T B^{-1}\mathbf{u} \neq \mathbf{0}.$$

Тогда матрица $A = B^{-1} + \mathbf{v}^T\mathbf{u}$ является обратимой и её обращение находится как

$$A^{-1} = B^{-1} - r^{-1}B^{-1}\mathbf{u}\mathbf{v}^T B^{-1}. \tag{1}$$

Далее будем считать, что матрица A обратимая с $a_{ii} \neq 0$ и нет необходимости в перестановках при её обращении. В этом случае, при условии $\mathbf{u} = \mathbf{e}_k = [0 \dots 1 \dots 0]^T$, выражение (1) можно упростить полагая, что $B^{-1}\mathbf{e}_k = \mathbf{b}_k^*$, $\mathbf{b}_k^*$ — k-ый столбец матрицы B^{-1}. Тогда столбец $\mathbf{a}_j^*$ матрицы A^{-1} можно вычислить как

$$\mathbf{a}_j^* = \mathbf{b}_j^* - \frac{\mathbf{v}^T\mathbf{b}_j^*}{1 + \mathbf{v}^T\mathbf{b}_j^*}\mathbf{b}_k^*, \quad j = 1, 2, \dots, N. \tag{2}$$

По соотношениям (2) будем вычислять обращение матрицы A, принимая что $B = E$ и вычисляя на k шаге

$$\mathbf{u} = \mathbf{e}_k, \quad \mathbf{v}^T = \mathbf{v}^k = \mathbf{a}^k - \mathbf{e}_k^T, \tag{3}$$

исходя из

$$A = B + \sum_{k=1}^{N} \mathbf{v}^k.$$

Столбец матрицы на k шаге вычисляется из соотношения

$$\mathbf{a}_j^{(k)} = \mathbf{a}_j^{(k-1)} - \frac{\mathbf{v}^T \mathbf{a}_j^{(k-1)}}{1 + \mathbf{v}^T \mathbf{a}_k^{(k-1)}} \mathbf{a}_k^{(k-1)}, \quad j = 1, 2, \ldots, N, \tag{4}$$

где $A^{(0)} = E$ и для $k = N$ находится $\mathbf{a}_j^{(N)} = \mathbf{a}_j^*, j = 1, 2, \ldots, N$.

Алгоритм 1 Параллельный алгоритм пополнения

Задано: $\mathbf{a}, \mathbf{v} \in \mathbb{R}^N$ {вектора хранятся в памяти GPU}

$\mathbf{v}^T = \mathbf{a}^k - \mathbf{e}_k^T$ { вектор $\mathbf{e}_k \in \mathbb{R}^N$, k элемент которого равен 1, а все остальные 0}
for $k = 0 \to N$ **do**
  for $i = 0 \to N$ **do**
    for $j = 0 \to N$ **do**
      $\beta_1 \leftarrow (\mathbf{v}_k^T, \mathbf{a}_j^{(k-1)})$ {вычисляется с помощью функции cublasDdot}
      $\beta_2 \leftarrow 1 + (\mathbf{v}_k^T, \mathbf{a}_k^{(k-1)})$ {cublasDdot}
      $\beta_3 \leftarrow \beta_1 / \beta_2$
      $\mathbf{a}_j^{(k)} \leftarrow \mathbf{a}_j^{(k-1)} - \beta_3$ {cublasDaxpy}
    end for
  end for
end for

В отличии от [6], где был исследован блочный алгоритм Шермана-Моррисона для плотных матриц, в котором требуется достаточно затратное разделение матриц на блоки, в рассматриваемом алгоритме предполагается распараллеливание скалярных произведений на GPU. Векторные операции распараллелены с помощью функций библиотеки cuBLAS.

3. Численные исследования

Численные эксперименты, представленные в таблице 1, были проведены на тестовых матрицах из известных библиотек [7,8], а также на матрицах полученных при решении задачи напряженно-деформированного состояния винтовой пружины методом дополнения Шура [3]. Незаполненные поля таблицы соответствуют вычислительным затратам существенно превышающим результаты по другим методам. Все матрицы симметричные и положительно-определённые. Размеры матриц варьируются от 276 до 18000, число ненулевых элементов — от 1666 до 6897316, числа обусловленности χ — от 10 до 10^{10}, а все результаты получены с двойной точностью.

Для вычислений использовался узел с четырехядерным процессором Intel Xeon CPU E5430 частотой 2667 МГц и графическим ускорителем NVIDIA GeForce GTX 580. Количество ядер CUDA 512, объем памяти 3 ГБ, частота ядра/памяти 772 МГц/4008 МГц.

Выделим три основных параметра для сравнения и оценки алгоритмов обращения матриц на параллельной вычислительной системе: число арифметических операций, затраты памяти и требования к обработке данных.

Затраты памяти метода сопряженных градиентов и метода пополнения на формирование исходной разреженной и обратной матриц равны $16NNZ + 4N^2$ и $8N^2$ байт, соответственно. На вспомогательные переменные, в методе пополнения, требуется $16N$, в методе сопряженных градиентов затраты несколько больше и составляют $48N$. Доступ к элементам матрицы в формате CSR осуществляется: по столбцам в методе SM, по строкам в CG, тогда как в методах Гаусса-Жордана и LU - факторизации доступ к матрице происходит поэлементно и на каждом шаге необходимо иметь всю матрицу.

Вычислительные затраты алгоритмов по числу операций выглядят следующим образом: в методе сопряженных градиентов без предобуславливания на каждой k-итерации

один раз выполняется матрично-векторное произведение и три скалярных произведения $N(N^2+3N+3)$; в методе пополнения на каждой итерации N раз выполняется два скалярных произведения $N^2(2N+2)$.

Для выполнения скалярных произведений в методе SM создается вектор в памяти GPU, в который записывается столбец матрицы, необходимый на данном шаге алгоритма. В матрично-векторном произведении для метода CG каждую строку матрицы обрабатывает несколько нитей (до 32) одновременно, что обеспечивает его выполнения в два и, даже, три раза быстрее скалярного произведения при произвольных размерах матриц и их разрежённости.

Определяющей, при использовании метода сопряженных градиентов, становится обусловленность (χ) обращаемой матрицы. В случае с хорошо обусловленными матрицами (когда число итераций метода сопряженных градиентов меньше размера матрицы), число операций в методе сопряженных градиентов будет меньше, чем в методе пополнения, так как при использовании метода пополнения требуется $2N$ скалярных произведений (см. табл. 1).

Таблица 1. Время обращения матриц (сек.)

			GJ	*LU*	CG	SM	
Название	N/Nnz	χ	CPU	CPU	GPU	CPU	GPU
Schur276	276/7488	1.2e+02	**0.1**	0.1	1.1	0.1	4.1
Lect 494_BUS	494/1666	3.9e+06	**0.3**	0.7	23.9	1.31	12.1
1138_BUS	1138/4054	8.0e+09	**9.5**	11.9	133.1	23.0	62.5
Schur1236	1236/40806	1.5e+01	12.6	18.6	**8.8**	34.5	74.5
BCSSTK11	1473/34241	5.3e+08	**21.4**	33.9	246.6	57.9	104.8
BCSSTK15	3948/117816	8.0e+09	426.0	649.1	**329.1**	1182.9	751.1
Shur5688	5688/203238	2.9e+01	1262.8	1822.5	**84.8**	3171.6	1572.2
ND3K	9000/3279690	1.6e+07	5009.6	7175.4	34801.3	12960.3	**3918.7**
msc10848	10848/1229776	9.9e+09	8718.8	12446.8	15259.6	21663.5	**5741.0**
Dubcova1	16129/253009	9.9e+02	29868.9	40859.7	**617.3**	75472.2	13006.5
bodyy4	17546/121550	8.1e+02	40400.6	—	**376.4**	—	15566.6
ND6K	18000/6897316	1.5e+07	41782.0	35674.5	—	—	**16386.0**

При плохой обусловленности матриц, но большой разреженности (Dubcova1 16129/253009, bodyy4 17546/121550) CG работает на порядок быстрее, чем метод SM. С увеличением числа ненулевых элементов затраты увеличивается (ND3K 9000/3279690, ND6K 18000/6897316), а девятикратное ускорение достигается в параллельном методе SM.

В численных экспериментах, для выполнения базовых операций рассматриваемых методов, вызывались функции библиотеки cuBLAS (см. Алгоритм 1). Особенностью применения функции `cublasDdot` является то, что при использовании текстурной памяти в основной

программе последующий вызов функции `cublasDdot` выполняется существеннее медленнее, чем другие вызовы при вычислении скалярных произведений. С увеличением размерности векторов это влияние становится значительным и операция скалярного произведения выполняется, практически, в семь раз медленнее, что характерно для метода CG, при реализации которого текстурная память задействована в матрично-векторном произведении.

Используемая реализация алгоритма CG хорошо масштабируется на GPU и позволяет достигать ускорения в десятки раз, по сравнению с вариантом, реализованном на CPU [1], а параллельная реализация алгоритма SM на GPU обеспечивает гораздо меньшие ускорения. Так, для матриц малых размерностей (например, матрицы Schur276, 494_BUS, 1138_BUS, Schur1236, BCSSTK11) время выполнения варианта, реализованного на GPU, больше, чем на CPU, а на матрицах больших размерностей получено максимальное шестикратное ускорение. Прежде всего это связано с лучшей масштабируемостью скалярного произведения небольших векторов на CPU, а не на GPU. В случае метода сопряженных градиентов значительное ускорение вычислений на GPU обеспечивает матрично-векторное произведение, которое обладает большой степенью параллелизма.

При использовании GPU на малых задачах (до $N = 3000$) сокращение времени выполнения операций не покрывают затраты на выделение памяти, поэтому эффективность параллельных реализаций на CPU в два раза выше. Однако, с увеличением размеров матриц, алгоритмы реализованные на CPU не показывают значимых ускорений, а результаты полученные на GPU, методами сопряженных градиентов и пополнения, эффективнее более чем в десять раз (например bodyy4).

Дальнейшие развитие высопроизводительных методов обращения матриц связано с модификацией и оптимизацией алгоритма метода пополнения при обращении симметричных матриц, в том числе и для построения эффективных предобуславливателей на его основе для метода сопряженных градиентов.

Литература

1. Копысов С. П., Кузьмин И. М., Недожогин Н. С., Новиков А. К. Параллельные алгоритмы формирования и решения системы дополнения Шура на графических ускорителях // Ученые записки Казанского университета. Серия Физ.-мат. науки – 2012. – Т. 154. №3. – С. 202-215.

2. Ezzatti P., Quintana-Orti E.S., Remon A. Using graphics processors to accelerate the computation of the matrix inverse // J. of Supercomputing. – 2011. V.58. – P.429-437.

3. Kopysov S. P., Kuzmin I. M., Nedozhogin N. S., Novikov A. K., Sagdeeva Y. A. Hybrid Multi-GPU solver based on Schur complement method // Lecture Notes in Computer Science – 2013. – vol. 7979. – pp. 65-79.

4. Фаддеев Д.К., Фаддеева В.Н. Вычислительные методы линейной алгебры. – М.: Физматгиз, 1963.

5. Sherman J., Morrison W.J. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix // Ann. Math. Statistics. – 1950. – V. 21. №1. – P. 124-127.

6. He X., Holm M., Neytcheva M. Efficiently parallel implementation of the inverse Sherman-Morrison algorithm //Lecture Notes in Computer Science. – 2013. – V. 7782. — P. 206-219.

7. Matrix Market: URL: http://math.nist.gov/MatrixMarket/ (дата обращения: 29.08.2013)

8. Tim Davis: University of Florida Sparse Matrix Collection: sparse matrices from a wide range of applications: URL: http://www.cise.ufl.edu/research/sparse/matrices/ (дата обращения: 29.08.2013)