УДК 004.021:519.683:519.684.6

О решении СЛАУ методом итераций Чебышёва на графических процессорах

Чадов С.Н., асп.

Рассматривается один из относительно редко используемых на практике итерационных методов решения СЛАУ – метод итераций Чебышёва. Приводится описание алгоритма, рассматриваются его преимущества для реализации на современных массивно-параллельных вычислительных системах. Рассматривается реализация алгоритма на графических процессорах NVIDIA, приводится анализ производительности, сравнение с реализацией аналогичного метода на центральном процессоре, а также с реализацией метода BiCG-Stab на GPU.

Ключевые слова: разреженные СЛАУ, GPGPU, CUDA, итерации Чебышёва.

On GPU-based Solving of Linear Systems with Chebyshev Iterations

S.N. Chadov, Post-Graduate Student

This article presents a study of one of rather seldom iterative linear methods, namely the Chebyshev iterations. The description of the algorithm is given, the advantages in massively parallel environment are discussed. The method is implemented on NVIDIA GPGPU, its performance is analyzed and compared against the similar method on CPU and a BiCG-STAB implementation.

Keywords: Sparse linear solver, GPGPU, CUDA, Chebyshev iterations.

Введение. Задача решения систем линейных алгебраических уравнений (СЛАУ) является одной из наиболее часто встречающихся при математическом моделировании различных систем. СЛАУ могут как описывать поведение некоторой модели сами по себе, так и возникать в ходе численного решения, например, системы дифференциальных уравнений в частных производных. С ростом размерности задач повышаются требования к производительности численных алгоритмов. Одним из наиболее востребованных направлений в современном численном моделировании является исследование итерационных методов для решения СЛАУ, имеющих ряд следующих преимуществ перед традиционными или, как их еще называют, «прямыми» методами, такими как LU-разложение:

- относительная простота и эффективность параллельной реализации;
- возможность эффективной работы с разреженными матрицами, что на порядок поднимает максимальную размерность решаемых задач.

Примерами достаточно хорошо известных эффективных итерационных методов являются методы сопряженных градиентов, GMRES, BiCG-Stab и др. [1].

С распространением технологий вычислений общего назначения на графических процессорах (GPGPU – general purpose graphical processor units) появились исследования, посвященные реализации итерационных методов с их использованием (например, работы [2], [3]). Мы рассмотрим достаточно редко применяемый на практике, в силу ряда присущих особенностей, итерационный метод, известный как метод итераций Чебышёва, и покажем, что этот метод имеет также ряд достоинств для реализации на GPGPU. В качестве технологии выберем NVIDIA CUDA.

Метод итераций Чебышёва. Итак, имеется СЛАУ вида

$$Ax = b,$$

где A – разреженная матрица большой размерности. Метод итераций Чебышёва представляет собой нестационарный итерационный метод, подходящий для решения как симметричных, так и несимметричных систем. Мы, однако, ограничимся симметричным случаем, что позволит существенно упростить программный код. Подробное описание метода можно найти в [4] и [1], мы лишь отметим основные его особенности и приведем псевдокод (рис. 1).

Дано: A, b – система уравнений;
l_{max} и l_{min} – аппроксимации собственных чисел;
M – матрица предобусловливания.

Вычислить $r = b - Ax^{(0)}$ для некоторого начального приближения $x^{(0)}$

$$c = \frac{l_{max} - l_{min}}{2}, \quad d = \frac{l_{max} + l_{min}}{2},$$

for i = 0, 1, …
  решить $Mz = r$
  if(i == *0*)
    $p = z$
    $\alpha = 2/d$
  else
    $\beta = \alpha(c/2)^2$
    $\alpha = \frac{1}{d - \beta}$
    $p = z + \beta p$
  endif
    $x = x + \alpha p$
  $r = b - Ax$
  проверить сходимость
end

Рис. 1. Псевдокод метода итераций Чебышёва

Основной особенностью метода итераций Чебышёва, ограничивающей его применимость, является необходимость априорного знания некоторой аппроксимации спектра матрицы *A*, причем с уменьшением точности этой аппроксимации сходимость алгоритма в существенной степени замедляется. Для симметричных положительно определенных матриц задача поиска такой аппроксимации упрощается: так как все собственные числа таких матриц действительны, для работы алгоритма достаточно двух констант l_{max} и l_{min}, обозначающих соответственно максимальное и минимальное собственные значения *A*. Далее рассмотрим некоторые подходы к решению этой проблемы, а пока будем полагать, что границы собственных значений матрицы нам известны.

Как видно из псевдокода (рис. 1), каждая итерация алгоритма требует две операции AXPY (ax + y), одно умножение матрицы на вектор (spMv) и одно вычисление предобусловливателя. Существенным отличием от остальных распространенных итерационных методов является отсутствие вычисления скалярных произведений, что является несомненным преимуществом для массивно-параллельных вычислительных систем, поскольку вычисление скалярного произведения требует операции редукции (reduce) над всеми вычислительными узлами, что для массивно-параллельных систем может существенно ограничивать производительность, а также существенно усложнить программирование (см., например, работу [5], полностью посвященную реализации редукции на GPU NVIDIA).

Реализация на GPGPU. Приведенную на рис.1 процедуру несложно запрограммировать с использованием библиотеки CuBLAS, реализующей основные операции линейной алгебры с использованием графических процессоров NVIDIA. Из необходимых для реализации операций в CuBLAS отсутствуют только функция вычисления предобусловливателя и функция умножения разреженной матрицы на вектор. В качестве предобусловливателя будем использовать метод Якоби, т. е. матрица *M* будет представлять собой диагональ матрицы *A,* что сводит шаг решения системы предобусловливателя к тривиальной операции поэлементного умножения векторов. Отметим, что метод Якоби является далеко не лучшим выбором с точки зрения скорости сходимости, однако более сложные методы (неполное LU-разложение, методы типа approximate inverse и т.д) представляют известные трудности при реализации на GPU.

Для умножения разреженной матрицы на вектор воспользуемся результатами работы [3], что означает, что матрица *A* будет представлена в формате CSR (compressed row storage). Контроль сходимости будем производить путем сравнения нормы вектора *r* с некоторой заранее определенной константой ε.

Такая «наивная» реализация успешно справляется с решением систем и показывает некоторое преимущество относительно аналогичной реализации на центральном процессоре, однако не в полной мере пользуется рассмотренными выше особенностями алгоритма для повышения производительности. Профилировка программы демонстрирует (рис. 2), что большую часть времени графический процессор простаивает в силу латентности вызова функций ядра (подробнее об особенностях выполнения программ на GPU NVIDIA см. [6]).

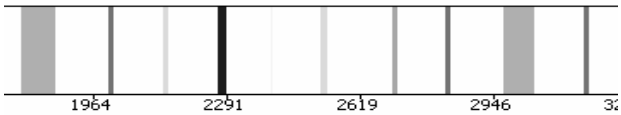

Рис. 2. Участок временного графика выполнения «наивной» реализации итераций Чебышёва (время простоя оборудования обозначено белым)

Во-первых, выбрав метод с отсутствием скалярных произведений, мы все равно привнесли операцию редукции для вычисления нормы и, соответственно, контроля сходимости. Можно попробовать использовать другой критерий сходимости, однако мы поступим по-другому: воспользуемся тем фактом, что итерации метода достаточно «легкие», и будем проверять сходимость не на каждой итерации, а, допустим, на каждой шестнадцатой итерации. В этом случае количество итераций возрастет не более чем на 15, а количество операций редукции снизится в 16 раз и не будет составлять сколько-нибудь существенной доли общего времени выполнения.

Во-вторых, ввиду отсутствия необходимости синхронизации, мы можем реализовать на графическом процессоре весь шаг алгоритма, вплоть до вычисления *r*. Такой метод показывает значительно большую производительность по сравнению с «наивным».

Можно пойти еще дальше и объединить в одном ядре все операции шага, кроме контроля сходимости. Однако такое решение имеет ряд особенностей. Как можно видеть, перед умножением матрицы на вектор в алгоритме нужна точка синхронизации (барьер). Это обусловлено тем, что функция умножения матрицы на вектор неизбежно использует другое разделение данных на потоки исполнения, нежели функции работы с векторами. Соответственно, любой поток может обратиться к данным, обрабатываемым другим потоком еще незавершенной операции BLAS. Однако создание глобального барьера на GPGPU сопряжено с большими сложностями. Мы можем попробовать проигнорировать требование глобальной синхронизации, ограничившись синхронизацией внутри каждого блока потоков. В таком случае мы получим стохастический (иногда в таком случае говорят *хаотический*) алгоритм, сходимость которого будет

иметь в себе элемент случайности. Такой подход не является чем-то необычным, в литературе (см., например, [7]) встречается описание стохастического метода типа Гаусса-Зейделя, в котором ради увеличения степени параллелизма также была отброшена синхронизация. Несмотря на это, мы не можем однозначно рекомендовать стохастический вариант алгоритма к использованию, поскольку его работоспособность, на наш взгляд, нуждается в серьезном математическом обосновании.

Результаты. Для тестирования приведенных алгоритмов воспользуемся тестовыми матрицами из [2], а также некоторыми матрицами из Harwell-Boeing collection. Будем использовать следующее оборудование: центральный процессор Intel Core i5 (4 ядра) и видеокарту GeForce GTX 260 (216 потоковых процессоров). Будем сравнивать результаты «наивного», оптимизированного и стохастического алгоритмов, реализацию итераций Чебышёва на центральном процессоре, а также реализацию BiCG-Stab на GPU, взятую из [3]. Анализ полученных результатов (табл. 1, 2) показывает, что для матриц относительно небольшого размера использованные оптимизации дают немалый прирост производительности, тогда как для матриц большого размера этот прирост весьма мал. Это объясняется тем, что для больших матриц время выполнения наиболее затратной операции spmv достаточно велико по сравнению со временем простоя оборудования из-за неэффективного шаблона вызова функций (на уменьшение которого и были направлены наши оптимизации). Это подтверждается графиком распределения времени выполнения функций (рис. 3), где функция axmb_csr_krnl обозначает операцию $b - Ax$.

Таблица 1. **Результаты исполнения для матрицы 1 (39204 строк, задача машинной графики)**

Алгоритм	Количество итераций	Время
Chebyshev – naïve	869	265
Chebyshev – optimized	864	131
Chebyshev – stochastic	768	101
Chebyshev – CPU	869	1400
BiCG-STAB – GPU	41	49

Таблица 2. **Результаты исполнения для матрицы 2 (350000 строк, задача машинной графики)**

Алгоритм	Количество итераций	Время
Chebyshev – naïve	2747	2900
Chebyshev – optimized	2656	2311
Chebyshev – stochastic	1247	2111
Chebyshev – CPU	2747	10656
BiCG-STAB – GPU	86	250

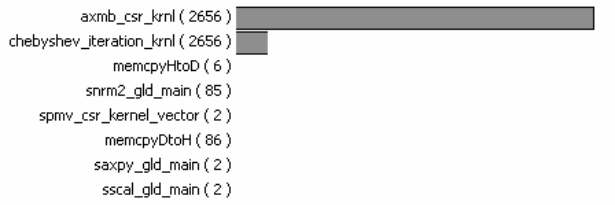

Рис. 3. График распределения времени выполнения алгоритма по функциям

Также можно отметить, что для матрицы большого размера наша реализация стохастического алгоритма не дала большого прироста производительности, несмотря на почти вдвое меньшее количество итераций. Это объясняется тем, что для поддержания вероятности сходимости алгоритма на приемлемом уровне мы были вынуждены использовать менее эффективную на нашем оборудовании скалярную реализацию spmv (о скалярных и векторных реализациях spmv см. [3]).

Также очевидно, что, по крайней мере при использованных нами значениях l_{max} и l_{min}, метод BiCG-Stab сходится значительно быстрее итераций Чебышёва. Однако при применении этого метода также возникают определенные сложности, например такие, как достаточно нерегулярная сходимость. Поэтому, на наш взгляд, метод итераций Чебышёва может быть использован в сочетании с другими методами, например BiCG-Stab, для уточнения решения. Такой подход частично исправляет и другой недостаток итераций Чебышёва, поскольку аппроксимацию собственных значений матрицы можно получить из результатов выполнения нескольких итераций по любому из методов сопряженных градиентов.

Список литературы

1. **Barrett R. et al.** Templates for the solution of linear systems: building blocks for iterative methods. – Philadelphia: SIAM, 1994.

2. **Buatois L., Caumon G., Lévy B.** Concurrent number cruncher: a GPU implementation of a general sparse linear solver // International Journal of Parallel, Emergent and Distributed Systems. – June 2009. – Volume 24. – Issue 3. – P. 205–223.

3. **Чадов С.Н.** Реализация алгоритма решения несимметричных систем линейных уравнений на графических процессорах // Вычислительные методы и программирование. – 2009. – №10. – С. 321–326.

4. **Golub G., Van Loan C.** Matrix Computations, 2nd ed. John Hopkins University Press. – Baltimore, 1989.

5. **Harris M.** Optimizing Parallel Reduction in CUDA. NVIDIA Tech. rep., 2008. URL: http://developer.download.nvidia.com/compute/cuda/1_1/Website/projects/reduction/doc/reduction.pdf

6. **NVIDIA.** CUDA Programming Guide. 2.1 edition, 2009.

7. **Chazan D., Miranker W.L.** Chaotic relaxation // Linear Algebra. – 1969. – Appl. 2. – P. 199–222.

Чадов Сергей Николаевич,
ГОУВПО «Ивановский государственный энергетический университет имени В.И. Ленина»,
аспирант кафедры высокопроизводительных вычислительных систем,
e-mail: sergei.chadov@gmail.com