Оптимизация расчётов в пакете OpenFOAM на GPU[1]

А. В. Монаков <amonakov@ispras.ru>

Аннотация. В статье рассматривается задача повышения скорости расчётов в пакете OpenFOAM за счёт переноса части вычислений на графические акселераторы (GPU). Приводится краткий обзор пакета и анализ переноса на GPU метода сопряжённых градиентов. Описаны несколько оптимизаций, часть из которых специфична для рассматриваемой реализации, а часть применима не только для GPU. Приводятся первичные результаты тестирования производительности.

Ключевые слова. OpenFOAM, CUDA, GPGPU, метод сопряжённых градиентов, предобуславливание на GPU, оптимизация для GPU

1. Введение

OpenFOAM — пакет библиотек и программ для организации научных расчётов преимущественно в области вычислительной гидродинамики и механики сплошных сред [1]. Пакет содержит поддержку запуска расчётов на кластере через MPI. Кроме приложений, выполняющих непосредственные расчёты, OpenFOAM содержит вспомогательные программы для подготовки сеток и постобработки результатов.

OpenFOAM содержит несколько десятков программ, выполняющих расчёты в соответствии с различными схемами. Выбор схемы зависит от особенностей решаемой задачи, таких как сжимаемость жидкости, ламинарность потока, способ учёта турбулентности и другие. Примерами этих программ в OpenFOAM являются icoFoam, pisoFoam, interFoam и другие. Обычно запуск этих программ требует наибольшего компьютерного времени в цикле использования OpenFOAM (подготовка сетки – решение – постпроцессинг и визуализация результатов) и поэтому требует тщательной оптимизации.

[1] Работа проводится в рамках реализации ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы», ГК № 07.514.11.4119 от 02 ноября 2011 года.

Работа проводится при поддержке программы «Университетский кластер», http://www.unicluster.ru

Большинство этих солверов в процессе своей работы решают множество дифференциальных уравнений на сетке, сводя их к обыкновенным системам линейных алгебраических уравнений над вещественными числами. OpenFOAM содержит лишь небольшое число методов решения СЛАУ. В то же время, для больших задач время решения этих систем может покрывать существенную долю общего времени расчёта. Таким образом, оптимизация методов решения СЛАУ является первоочередным способом повышения производительности OpenFOAM.

OpenFOAM содержит реализации следующих методов решения СЛАУ:

- Метод сопряжённых градиентов, с предобуславливанием (PCG). Этот алгоритм применим для систем с симметричной матрицей (в контексте OpenFOAM это обычно системы для давления).
- Метод бисопряжённых градиентов (PBiCG). Вариант предыдущего метода, работающий для систем с несимметричной матрицей (например, уравнения для компонент скоростей). В развиваемом сообществе проекте OpenFOAM-extend [2] также реализован более надёжный стабилизированный метод бисопряжённых градиентов (BiCGStab). Как и PCG, эти реализации используют предобуславливание для увеличения скорости сходимости.
- Геометро-алгебраический многосеточный метод (GAMG). Этот метод применим вне зависимости от симметричности матрицы системы.

Все реализации поддерживают распределённое решение на кластере, используя коммуникации через MPI.

Все приведённые методы решения являются не прямыми (как, например, метод Гаусса для решения плотных систем), а итерационными: при решении системы $Ax = b$ они начинают с некоторого начального приближения x_0 и циклически строят новые решения x_1, x_2, ..., которые постепенно приближаются к точному решению. На практике достаточно близкие к точному решения получаются за сравнительно небольшое по сравнению с размером системы количество итераций.

Большинство несимметричных систем в OpenFOAM решается за сравнительно малое (обычно не более 10) количество итераций. Это означает, что использование акселератора с отдельной памятью не может существенно ускорить решение этих систем, так как возникнут накладные расходы, связанные с копированием матрицы системы и векторов в память акселератора. Таким образом, основное внимание требуют метод сопряжённых градиентов и многосеточный метод для случая симметричных систем.

Матрицы решаемых систем являются весьма разреженными. Поскольку в большинстве задач используется метод конечных объёмов с дискретизацией первого порядка, количество строк и столбцов в матрицах систем равняется количеству ячеек сетки, а количество внедиагональных ненулевых элементов

в каждой строке — количеству соседей соответствующей ячейки. В частности, для задач, решаемых на ортогональных трёхмерных сетках, большинство строк системы содержат семь ненулевых элементов.

2. Предобуславливание

Пусть для матрицы A известен оператор M, в каком-то смысле приближающий обратную к A матрицу. Тогда при решении системы

$$Ax = b$$

можно перейти к решению эквивалентной системы

$$AMx' = b,\ x = Mx'$$

В силу выбора M, матрица новой системы AM близка к единичной, и для неё итерационные методы будут сходиться быстрее. M называется предобуславливающим оператором. Известно множество подходов к построению предобуславливателей, некоторые из которых будут рассмотрены далее [3].

3. Постановка задачи

Нашей задачей является доработка OpenFOAM с целью повышения производительности решателей за счёт использования GPU. В соответствии с обзором, это включает решение следующих подзадач:

1. Портирование итерационных методов решения на GPU для работы на одном GPU:
   1. Метода сопряжённых градиентов (PCG)
   2. Метода бисопряжённых градиентов (PBiCG, PBiCGStab)
   3. Многосеточного метода
2. Поддержка запуска на кластере, содержащим несколько узлов с GPU:
   1. Поддержка MPI-коммуникаций в GPU-варианте солвера
   2. Выбор метода поддержки использования нескольких GPU на одном узле

Описываемая в статье работа была посвящена портированию и оптимизации метода сопряжённых градиентов на GPU. Несимметричные солверы (1.2) не были портированы, так как на всех тестовых задач они сходятся достаточно быстро, и порт на GPU не даст заметного прироста производительности.

4. Реализация метода сопряжённых градиентов на GPU

Простой вариант метода сопряжённых градиентов выглядит в псевдокоде следующим образом [4]:


```
r = b - Ax
p = z = Mr
v' = (r, z)
do {
  t = Ap
  a = v' / (p, t)
  x = x + ap
  r = r - at
  if (small (r))
        break;
  z = Mr
  v = (r, z)
  w = v / v'
  v' = v
  p = r + wp
}
```

(где A — матрица системы, M — выбранный предобуславливатель, b — правая часть, x — начальное приближение и искомое решение, r, z, p, t — вспомогательные векторы, v, v', a, w — скалярные величины, (,) - взятие скалярного произведения).

Таким образом, на GPU необходима реализация следующих операций:

1. Простые операции уровня BLAS-1: взятие линейной комбинации и скалярного произведения.
2. Умножение разреженной матрицы на вектор.
3. Применение предобуславливающего оператора.

Все эти операции будут ограничены на GPU пропускной способностью памяти. Соответственно, минимизация трафика в глобальной памяти GPU важна для повышения скорости GPU-реализации, а достигнутая пропускная способность может использоваться для оценки эффективности.

Первая группа операций несложно распараллеливается на GPU и доступна в библиотеке CUBLAS. Умножение матрицы на вектор исследовано в множестве работ. Известно, что эффективность ядра на GPU существенно зависит от формата, в котором представлена разреженная матрица. В нашей работе используется разработанный ранее специализированный формат и ядро умножения, которые позволяют получить более высокую по сравнению с опубликованными ранее подходами производительность.

Отметим, что в случае однопоточной работы копирование данных через интерфейс PCI-Express внутри цикла итерационного метода необходимо только в одном месте: при использовании нормы вектора невязки для выхода из цикла. В остальных случаях можно хранить все промежуточные значения исключительно на GPU, что требует реализации тривиального ядра взятия

отношения двух чисел (и противоположного ему, для операции r = r - at). В нашей реализации ненужные копирования исключены, а копирование нормы невязки осуществляется асинхронно.

В случае распараллеленного запуска возникнут дополнительные копирования через сеть после операций взятия скалярного произведения и перед домножением на матрицу системы.

5. Предобуславливание на GPU

5.1. Вычисление предобуславливателя

Задача эффективного предобуславливания на GPU исследована сравнительно неглубоко. Известны две основные категории предобуславливателей:

1. Основанные на неполном разложении на треугольные множители (ILU). Эти предобуславливатели могут быть вычислены сравнительно дёшево, но их применение требует решения двух треугольных систем уравнений, что обладает низким параллелизмом и сложно реализуемо для GPU.
2. Основанные на явном вычислении приближённой обратной матрицы (FSAI, AINV). Применение этих предобуславливателей является простым умножением матрицы на вектор, однако их вычисление сравнительно дорого.

Также следует отметить, что существуют блочные варианты ILU, разработанные для многопроцессорных реализаций, и простой диагональный предобуславливатель, который хорошо подходит для GPU и не требует долгих предвычислений, но слабо снижает скорость сходимости итерационного метода.

Для реализации в рамках первого этапа был выбран предобуславливатель AINV [5]. Он хорошо подходит для ранних исследований, так как параметризован drop tolerance — минимальным абсолютным значением своих ненулевых элементов, варьирование которого позволяет тестировать различные компромиссы между скоростью применения предобуславливателя и общей скоростью сходимости.

Параметр drop tolerance (droptol) может быть выбран в диапазоне 0.0 .. 1.0. Для droptol = 0 AINV построит точное разложение исходной матрицы на треугольные множители. В этом случае метод сопряжённых градиентов сойдётся за одну итерацию (если пренебречь потерями точности в вещественной арифметике), но полученные множители будут плотно заполненными треугольными матрицами, т.е. умножение на предобуславливатель будет слишком дорогой операцией, а количество памяти, необходимой для его представления, будет пропорционально квадрату размера системы.

Для droptol = 1.0 полученные треугольные множители, как правило, не будут содержать ненулевых элементов вне диагонали. Таким образом, полученный предобуславливатель будет совпадать с диагональным: его применение будет сравнительно дешёвой операцией, но скорость сходимости итерационного метода может оказаться слишком низкой.

Идеальное значение drop tolerance зависит от решаемой системы, и динамический подбор оптимального значения этого параметра является нетривиальной задачей.

Отметим, что существует возможность полностью скрыть расходы, связанные с вычислением предобуславливателя. Она основана на том, что последовательно решаемые для одного и того же дифференциального уравнения системы имеют близкие коэффициенты; как следствие, получаемые для них предобуславливатели также будут мало отличаться. Поскольку точное знание предобуславливателя не требуется, возникает возможность повторно использовать предобуславливатели, полученные для предшествующих матриц. Это соображение позволяет вычислять предобуславливатель не каждый раз, а только для какой-то доли решаемых систем.

Развивая эту идею дальше, можно заметить, что вычисление предобуславливателя можно организовать асинхронно. Таким образом, каждый запуск реализации метода сопряжённых градиентов на GPU может немедленно начинать итерации на GPU, используя последний из полученных ранее предобуславливателей, и в то же время запускать вычисление предобуславливателя для текущей системы в отдельной нити на CPU (если такая нить не активна).

5.2. Применение предобуславливателя

Применение предобуславливателя является самой дорогой операцией в цикле метода сопряжённых градиентов (если не принимать во внимание случаи близких к единице значений drop tolerance). Были найдены два способа оптимизации:

1. Использование формата с одинарной точностью (float) для хранения предобуславливателя. Это возможно сделать, так как для предобуславливателя потеря точности не важна.
2. Переупорядочивание строк и столбцов в треугольных множителях, которыми оперирует AINV.

Последний пункт требует отдельного пояснения. Дело в том, что полученный по алгоритму AINV предобуславливатель представляет собой произведение верхнетреугольной и нижнетреугольной матрицы (для симметричной системы эти матрицы получаются друг из друга транспозицией). Таким образом, применение предобуславливателя выполняется как два домножения вектора невязки на матрицу: сначала на нижнетреугольную, затем на верхнетреугольную. Было замечено, что домножение на нижнетреугольный

множитель требует в два-три раза больше времени на GPU, хотя количество ненулевых элементов в них одинаково. Проще всего такой эффект объясняется тем, что блоки CUDA-нитей на GPU запускаются в порядке возрастания их индексов; таким образом, для нижнетреугольного множителя сначала будут запущены блоки, обрабатывающие слабо заполненные верхние строки матрицы. Это может приводить к тому, что более плотно заполненные нижние строки обрабатываются блоками, запущенными в последнюю очередь, что приводит к неполной загрузке GPU.

Поскольку промежуточный вектор, полученный от домножения на нижнетреугольный множитель, используется исключительно для последующего домножения на второй множитель предобуславливателя, можно заметить, что можно обратить порядок строк нижнетреугольного множителя и получать таким образом "перевёрнутый" промежуточный вектор. Чтобы сохранить результат применения предобуславливателя, далее достаточно обратить порядок столбцов второго, верхнетреугольного множителя. После этого оба множителя становятся лево-верхнетреугольными, и не приводят к эффекту плохой балансировки нагрузки на GPU.

6. Переупорядочивание расчётной сетки

Известно, что для фиксированного drop tolerance количество ненулевых элементов в множителях AINV существенно зависит от порядка нумерации элементов расчётной сетки. Это объясняется тем, что порядок нумерации влияет на структуру расположения ненулевых элементов ("портрет" разреженной матрицы), а от неё, в свою очередь, зависит количество добавленных ненулевых элементов в процессе биортогонализации, с помощью которого строится предобуславливатель.

Аналогично, для основанных на неполном треугольном разложении (ILU) предобуславливателей перестановка ячеек также имеет значение. Для ILU0 (ILU-разложение без добавления ненулевых элементов) она влияет на локальность по кешу при решении треугольных систем.

Порядок, который соответствует исходной нумерации ячеек полученной от какого-либо генератора сетки, обычно не оптимален с точки зрения минимизации заполнения предобуславливателя. Так, для ортогональной сетки в кубической расчётной области порядок будет соответствовать лексикографической нумерации ячеек. Этот порядок породит семидиагональную матрицу, а AINV добавит плотное облако элементов вокруг каждой диагонали.

В OpenFOAM реализовано переупорядочивание в соответствии с порядком поиска в ширину (упрощённый вариант алгоритма Cuthill-McKee [6, гл. 4.5]). Такое переупорядочивание хорошо подходит для ILU0-предобуславливателя, аналогом которого является используемый в OpenFOAM DILU, так как

группирует ненулевые элементы плотнее вокруг главной диагонали. Воспользоваться этой реализацией позволяет утилита renumberMesh.

В то же время, для AINV более предпочтительно использовать так называемые "снижающие заполнение" порядки (fill-reducing ordering). Нами была реализована библиотека для OpenFOAM, которая позволяет переупорядочивать сетки по алгоритму Nested Dissection (используется реализация из библиотеки METIS [7]).

7. Вычисления со смешанной точностью на GPU

Структура итерационных методов решения систем уравнений позволяет успешно использовать вычисления со смешанной точностью, когда для решения системы, заданной в двойной точности, большая часть вычислений производится с одинарной точностью, а небольшая доля вычислений с двойной точностью используется для того, чтобы обеспечить такую точность, как если бы использовалась реализация, использующая исключительно двойную точность.

Обычно в статьях, обсуждающих вычисления со смешанной точностью, отправной точкой для обоснования их использования является ограниченность ресурсов ALU двойной точности (ранние GPU или FPGA, в которых снижение количества ALU двойной точности позволяет разместить больше ALU одинарной точности на кристалле [8]). Однако даже для устройств, не испытывающих существенного недостатка в ALU двойной точности (GPU серии Tesla на технологии Fermi), использование этого подхода приносит существенное ускорение, так как вычисления ограничены пропускной способностью памяти.

Нами была реализована схема итеративного уточнения (iterative refinement) для метода сопряжённых градиентов с сохранением вектора направления спуска (p) и обновлениями вектора невязки, когда его норма снижается в 16 раз по сравнению с максимальной после последнего обновления решения в двойной точности (иными словами, когда получены 4 старших двоичных разряда вектора поправки, он прибавляется к вектору решения в двойной точности, и невязка пересчитывается также в двойной точности; вектор направления остаётся неизменным).

8. Экспериментальные результаты

Далее приведены результаты тестирования описанной реализации на узле с CPU Intel Xeon X5650 (2.67 GHz) и GPU Nvidia Tesla M2090. Распределённый запуск OpenFOAM не использовался, т.е. задействовано только одно ядро процессора.

8.1. Исследование ускорения в зависимости от размера сетки

В таблице 1 приведены результаты запусков для тестовой задачи Cavity3D. Задача получена из стандартного примера icoFoam/cavity путём преобразования геометрии области в трёхмерный куб. При уменьшении шага сетки в два раза шаг по времени также уменьшался в два раза, чтобы сохранить сходимость схемы; при этом конечное время симуляции уменьшалось в 16 раз, чтобы приблизительно сохранить общее количество вычислений.

Сетка	Шаг	End time	Время на CPU, с	Время на GPU, с	Ускорение
20x20x20	0.001000	2.00000	22	42	0.52
40x40x40	0.000500	0.12500	71	56	1.27
80x80x80	0.000250	0.00775	284	91	3.12
100x100x100	0.000125	0.00050	105	48	2.18

Таб. 1. Результаты экспериментальных запусков OpenFOAM.

При решении на CPU использовался метод PCG в двойной точности со стандартным для него предобуславливателем DILU. При решении на GPU использовалась фиксированный drop tolerance, равный 0.1; перенумерация сетки и вычисления со смешанной точностью не использовались. В столбцах «Время на CPU/GPU» указано общее время счёта в icoFoam.

Можно отметить, что из-за накладных расходов GPU реализация медленнее в два раза для сетки с 8000 ячеек, но уже для 64000 ячеек наблюдается прирост производительности.

9. Основные результаты

В качестве основных результатов работы хотелось бы отметить реализацию следующих оптимизаций:

1. Асинхронное вычисление предобуславливателя.
2. Использование лево-верхнетреугольных множителей в AINV.
3. Использование переупорядочивания расчётной сетки для повышения эффективности предобуславливания.
4. Использование вычислений со смешанной точностью.

Насколько известно автору, первые два улучшения предложены впервые.

10. Заключение

В дальнейшем планируется добавить поддержку распределённого запуска в разработанную реализацию метода сопряжённых градиентов. Также планируется разработать и протестировать GPU версии многосеточного метода и BiCGStab. Наконец, необходимо найти эффективный метод автоматического подбора drop tolerance для предобуславливателя.

Кроме того, интересно будет исследовать возможность более эффективного по сравнению с распараллеливанием через MPI использования нескольких акселераторов на одном узле, возможность использования метода IDR(s), который может обладать более высокой скоростью сходимости, оптимизацию вычисления предобуславливателя и перенумерации сетки под особенности GPU.

Список литературы

[1] SGI, The OpenFOAM Foundation, http://openfoam.org/
[2] The OpenFOAM Extend Project, http://www.extend-project.de/
[3] M. Benzi, Preconditioning techniques for large linear systems: a survey. J. Comput. Phys., 128 (2002), 418–477
[4] Y. Saad, Iterative methods for sparse linear systems, SIAM, Philadelphia, 2003, 567
[5] R. Bridson, W.-P. Tang, Refining an approximate inverse, Journal on Computational and Applied Math, 123 (2000), Numerical Analysis 2000 vol. III: Linear Algebra, pp. 293-306.
[6] S. Pissanetzky, Sparse Matrix Technology, Academic Press, Waltham, 1984, 312
[7] G. Karypis, V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system, version 4.0, http://www.cs.umn.edu/~metis, 2009
[8] D. Göddeke, R. Strzodka, S. Turek, Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations, International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), Special issue: Applied parallel computing, 22 (2007), 221–256

Optimizing OpenFOAM GPU Solvers

Alexander Monakov amonakov@ispras.ru, ISP RAS, Moscow, Russia

Abstract.The paper presents preliminary research on improving performance of CFD simulations in OpenFOAM via offloading parts of computations (specifically, solution of linear systems) to a graphics accelerator (GPU). We present a short review of OpenFOAM package and describe porting conjugate gradient method to the GPU architecture using CUDA programming model. Porting the basic algorithm is straightforward, however care should be taken to avoid unnecessary copying over PCI-Express bus. Efficient preconditioning on the GPU is then discussed. We use approximate inverse preconditioning, which can be implemented with good parallelism on the GPU. To amortize the cost of preparing the preconditioner, we allow reuse of preconditioners on the GPU and compute them on the CPU in a helper thread asynchronously. We mention several optimization opportunities: reordering the preconditioner to upper-left triangular form so that CUDA blocks multiplying by denser parts of preconditiner factors are scheduled first; using single-precision storage for the preconditioner to save memory bandwidth; reordering the mesh with nested dissection method from Metis library and using mixed-precision iteration for the conjugate gradient method. Preliminary performance testing results show performance improvement starting from 64000-cell meshes and reaching 2x for a 1-million cell mesh for a non-parallel run. As future work we mention support for parallel runs with MPI, research of other solvers such as multigrid, BiCGStab and IDR, and choosing drop tolerance automatically for the AINV preconditioner.

Keywords: OpenFOAM, CUDA, GPGPU, conjugate gradient technique, preconditioning in GPU, optimization for GPU

References

[1]. SGI, The OpenFOAM Foundation, http://openfoam.org/
[2]. The OpenFOAM Extend Project, http://www.extend-project.de/
[3]. M. Benzi, Preconditioning techniques for large linear systems: a survey. J. Comput. Phys., 128 (2002), 418–477
[4]. Y. Saad, Iterative methods for sparse linear systems, SIAM, Philadelphia, 2003, 567
[5]. R. Bridson, W.-P. Tang, Refining an approximate inverse, Journal on Computational and Applied Math, 123 (2000), Numerical Analysis 2000 vol. III: Linear Algebra, pp. 293-306.
[6]. S. Pissanetzky, Sparse Matrix Technology, Academic Press, Waltham, 1984, 312
[7]. G. Karypis, V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system, version 4.0, http://www.cs.umn.edu/~metis, 2009
[8]. D. Göddeke, R. Strzodka, S. Turek, Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations, International Journal of Parallel, Emergent and Distributed Systems (IJPEDS), Special issue: Applied parallel computing, 22 (2007), 221–256