СРАВНЕНИЕ СКОРОСТИ РАБОТЫ GPU И CPU ПРИ РЕШЕНИИ ЗАДАЧИ КЛАССА «DATA INTENSIVE»

М.А. Черноскутов

Введение

В последнее время растет популярность использования ГПУ в качестве вычислителей общего назначения, с помощью которых потенциально можно запрограммировать решения самых разнообразных задач. Сейчас ГПУ используются в основном для приложений, которые принято называть вычислительно-интенсивными («computing intensive tasks»): механика сплошных сред, молекулярная динамика, трассировка лучей и т.д. Однако растет потребность и в задачах совершенно другой природы – «data intensive tasks», которые характеризуются работой с большими массивами данных и активным использованием нерегулярного доступа в память [1]. На данный момент уже появился специализированный рейтинг Graph500 (аналог рейтинга Top500) [2], целью которого является оценка производительности вычислительных систем при обработке больших массивов данных.

Современные вычислительные элементы, доступные для широкого рынка, такие как ГПУ и ЦПУ, предназначены для решения вычислительно-интенсивных задач, т. к. ориентированы на работу с хорошо структурированными наборами данных и обработку вещественных чисел. «Data intensive» задачи, напротив, требуют быстрого извлечения информации из огромных массивов данных (часто еще и неструктурированных) и целочисленную арифметику [2]. Однако ГПУ обладают большим количеством вычислительных модулей и широким каналом доступа в свою внутреннюю память, что может положительно сказаться на скорости параллельной обработки больших массивов данных. Следовательно, актуальной является задача оценки эффективности использования ГПУ при решении «data intensive» задач.

Выбор задачи класса «data intensive»

В качестве «data intensive» задачи выбран обход графа в ширину. Алгоритм обхода графа (как в ширину так и в глубину) заключается в разметке всех вершин в графе, начиная с одной заранее заданной, в определенном порядке. Более подробно об этом алгоритме можно прочитать в [3]. Он используется и в качестве ядра теста Graph500 (пока реализация доступна только для ЦПУ) [4]. Данный выбор обусловлен следующими обстоятельствами [2]:

- алгоритм обхода графа в ширину часто используется в реальных научных и технических приложениях (поиск сообществ в социальных сетях, перебор большого количества данных для нужд информационной безопасности крупных компаний, анализ месторождений при нефтедобыче)
- структура графов отображает реальные наборы данных, которые встречаются в задачах
- результаты масштабирования задачи обхода графов должны хорошо соответствовать масштабированию реальных приложений

Известно множество работ по обработке больших графов на ЦПУ и ГПУ и сравнению их производительности[5-9]. Например, в работе [5] предпринята попытка реализовать ряд алгоритмов на графах (в том числе и поиск в ширину) ан ГПУ с архитектурой Tesla. В работах [6–8] содержится более глубокий анализ пригодности архитектур ГПУ (в том числе и последнего семейства Fermi) для нужд «data intensive» задач. Работа [9] описывает использование библиотеки «Dislib» для модификации теста Graph500.

Постановка задачи

Целью работы является оценка эффективности использования ГПУ в задаче обхода графов в ширину. Сравнение проводится с многоядерным ЦПУ. Распараллеливание алгоритма обхода графа осуществляется по всем доступным ядрам ГПУ и ЦПУ. В качестве входных данных для алгоритма используются графы одинакового размера. Результатом работы программы является время и скорость обхода заданного графа (измеряется в количестве пройденных ребер в секунду). Ранжирование результатов осуществляется по скорости обхода графа (так же, как и в рейтинге Graph500 [2]).

Описание графов

В каждом эксперименте использовался ориентированный граф, с заранее заданным числом исходящих из каждой вершины ребер. При этом вершины на противоположных концах ребер выбирались случайным образом (не исключая возможности образования петель и несвязных компонент). Графы с подобной структурой часто встречаются в реальных задачах, таких как обработка социальных сетей (например, если представить в качестве вершин пользователей сети, а ребрами соединять каждую пару друзей). При этом они довольно просты в реализации и хорошо загружают вычислительную систему. Размер графа можно оценить довольно точно, т. к. при его генерации количество неоднозначностей сведено к минимуму (т. е. заранее известно количество вершин и количество ребер). Размер выбирался таким образом, чтобы заполнить всю доступную память ОЗУ на ГПУ. Используемые в работе графы приведены на табл.1.

Таблица 1. Описание используемых графов

Количество исходящих из вершины ребер, шт.	Количество вершин, шт.	Размер графа, ГБ
5	66060288	2,21
10	48234496	2,52
15	35651584	2,52
20	28311552	2,53
25	23068672	2,49
30	19922944	2,52
35	16777216	2,44
40	14680064	2,41
45	13631488	2,49
50	12582912	2,53
55	11534336	2,54
60	10485760	2,50
65	9437184	2,43
70	8388608	2,31

Алгоритм обхода графов

Для операции обхода графа на ГПУ используются два CUDA-ядра. Первое перебирает вершины на текущей итерации (в порядке обхода в ширину) и заносит данные об их соседях в специальный массив, а второе, анализируя этот массив, формирует новый список вершин для следующей итерации алгоритма. Разделение обхода вершин графа и формирования списка для новой итерации на два отдельных ядра сделано, прежде всего, с целью устранения неэффективных атомарных операций. Каждому потоку ставится в соответствие одна вершина. В текущей реализации в каждом блоке всегда задействовано по 1024 потока. Количество блоков варьируется в зависимости от количества исходящих ребер для каждой вершины: от 8192 для 70 исходящих ребер до 64512 для 5 ребер. Таким образом, каждой вершине назначается свой поток ГПУ. Shared-память ГПУ в реализации данного алгоритма не использовалась, т. к. шаблон доступа в память имеет произвольный характер (много одновременных доступов в несмежные ячейки).

Аналогично программе для ГПУ, для ЦПУ используется две функции, распараллеленные на все доступные ядра ЦПУ: первая обходит вершины на текущей итерации, а вторая формирует список вершин для следующей итерации. Из-за значительно меньшего числа ядер в ЦПУ, по сравнению с ГПУ, вершины обрабатываются блоками, количество которых равно количеству ядер в ЦПУ.

Аппаратное и программное обеспечение

Эксперименты проводились на материально-технической базе суперкомпьютерного центра Института математики и механики УрО РАН. Для тестирования использовалось ГПУ Nvidia Tesla M2050, имеющее 448 вычислительных ядер и 3 ГБ памяти ОЗУ (в действительности, доступным для использования оказалось около 2.5 ГБ из-за включенного ECC и других накладных расходов) и ЦПУ Intel Xeon X5675, имеющий 6 вычислительных ядер и 48 ГБ оперативной памяти. Оба вычислителя располагаются в сервере HP ProLiant SL390s G7 4U.

Программа для ГПУ скомпилирована с использованием nvcc, входящего в комплект CUDA Toolkit 4.0. Для ЦПУ использовался компилятор icc версии 11.1 и технология OpenMP. Для распраллеливания обхода графа в ширину на все процессоры в ЦПУ и потоки в ГПУ использовались технологии OpenMP и CUDA. Для создания графов использовался генератор случайных числе rand(), входящий в стандартную библиотеку языка C.

Описание и результаты эксперимента

Программам на ГПУ и ЦПУ подавались графы с одинаковым числом вершин и одинаковым количеством ребер, исходящих из каждой вершины. Эксперименты проводились с графами, имеющими от 5 до 70 исходящих из каждой вершины ребер. Количество вершин в графе варьировалось от 8388608 до 66060288, в зависимости от объема доступной памяти и числа исходящих из каждой вершины ребер.

Характерной особенностью алгоритма обхода графов является быстрый рост числа вершин, которых необходимо обработать на текущей итерации алгоритма. На рис.1 представлена зависимость количества вершин на каждой итерации от ее номера для графа размером 8388608 элемента.

Рис.1. Распределение количества обрабатываемых вершин в графе по итерациям

Разные графики соответствуют разному количеству исходящих из вершины ребер. Видно, что практически весь обход графа происходит на 2–3 итерациях – в этом месте графика появляется характерный пик. Таким образом, резко увеличивается число обращений в память как ГПУ, так и ЦПУ из-за потребности в обработке большого массива данных. Причем, при увеличении количества исходящих ребер, пик растет и смещается влево, сигнализируя о дополнительном росте нагрузки на каждой итерации, что вызывает деградацию производительности (рис.2).

Как видно из рис.3, ГПУ заметно опережает ЦПУ во всех экспериментах. Наибольшее преимущество ГПУ имеет при обходе графов с малым числом исходящих ребер. При увеличении числа исходящих ребер, преимущество ГПУ постепенно уменьшается, вследствие роста количества запросов в память, производимых множеством параллельных нитей. В ЦПУ, наоборот, параллельных потоков намного меньше и вычислительные ядра имеют большую скорость работы по сравнению с нитями ГПУ.

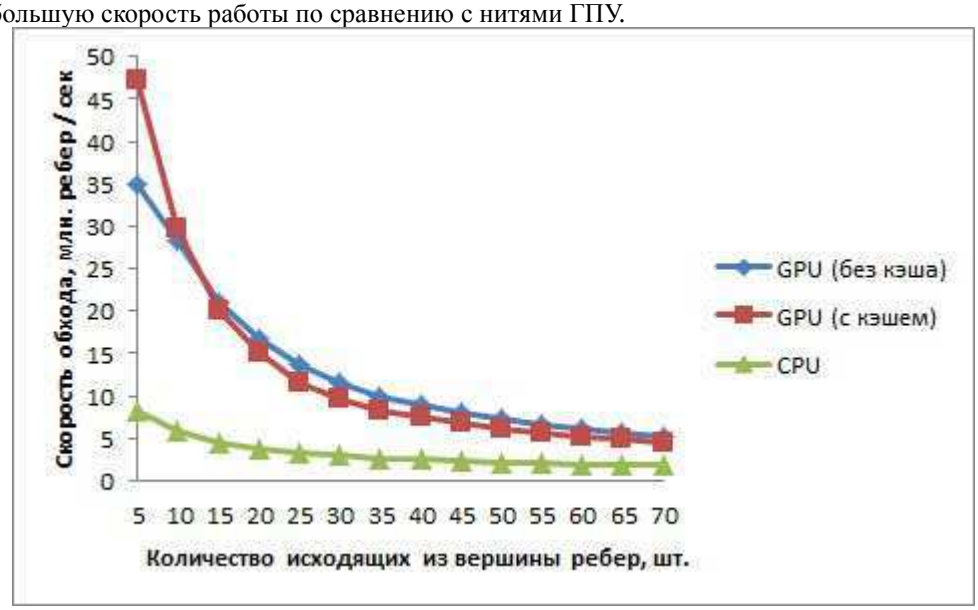

Рис.2. Зависимость скорости обхода графов на ЦПУ и ГПУ в зависимости от количества исходящих из каждой вершины ребер

Достигнутое на ГПУ ускорение показано на рис.3.

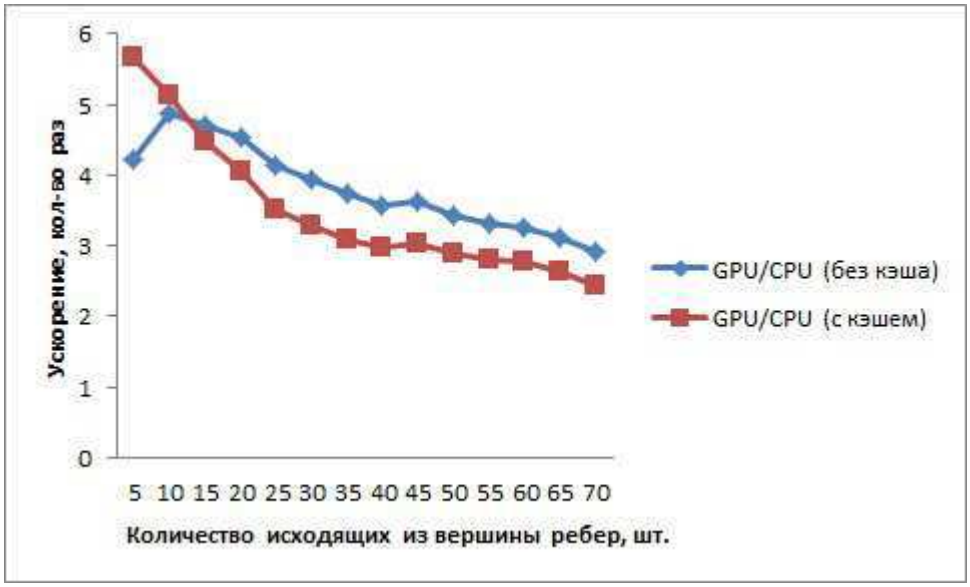

Рис.3. Ускорение, полученное на ГПУ

Интересной особенностью является влияние кэша L1 ГПУ на скорость обхода графа. При малом числе исходящих ребер кэш ГПУ оказывает положительное влияние на производительность, т.к. может захватить больше данных о вершинах графа, а с ростом их числа, наоборот, показывает ухудшающиеся результаты. К тому же с ростом количества исходящих ребер снижается пространственная и временная локальность [10] и степень промахов кэша из-за этого только увеличивается.

Выводы

В результате эксперимента выявлено, что обход графов на ГПУ производится с большей скоростью, по сравнению с аналогичной задачей ЦПУ, при условии, что размер графа не превышает объем памяти в ГПУ. Следовательно, ГПУ могут применяться для решения задач класса «data intensive».

В качестве дальнейших направлений исследований интерес представляет:

- использование для создания графов более реалистичных генераторов случайных чисел (таких как cuRand для ГПУ)
- обход графов, полученных с помощью генераторов графов (таких как Kronecker Graph Generator [11]), а также графов полученных в результате решения реальных задач (к примеру графы из Sparse Matrix Collection [12] или DIMACS Implementation Challenge [13])
- анализ эффективности обхода больших графов в мульти-ГПУ системах, а также в кластерных системах, использующих ГПУ

Работа поддержана грантом УрО РАН РЦП-12-П13

ЛИТЕРАТУРА:

1. Furht B., Escalante A. «Handbook of Data Intensive Computing», Springer, 2011.
2. Murphy R., Wheeler K., Barrett B., Ang J. «Introducing the Graph 500» // Cray User's Group (CUG), May 5, 2010.
3. Кормен Т., Лейзерсон Ч., Риверс Р., Штайн К. «Алгоритмы. Построение и анализ» – М.: «Вильямс», 2011.
4. Graph 500 Benchmark 1 (“Search”) // URL:http://www.graph500.org/specifications (дата посещения: 01.06.2011).
5. Harish, P. and Narayanan, P.J. «Accelerating large graph algorithms on the GPU using CUDA» // Proceedings of the 14th international conference on High performance computing (Berlin, Heidelberg, 2007), 197–208.
6. Hong, S. et al. «Accelerating CUDA graph algorithms at maximum warp» // Proceedings of the 16th ACM symposium on Principles and practice of parallel programming (New York, NY, USA, 2011), 267–276.
7. Merrill D., Garland M., Grimshaw A. «Scalable GPU graph traversal»// Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP’12), 117-128.
8. Hong S., Oguntebi T., Olukotun K. «Efficient Parallel Graph Exploration on Multi-Core CPU and GPU» // Proceedings of the 2011 International Conference on Parallel Architectures and Compilation Techniques (Galveston, TX, USA, October 10-14, 2011), 78–88.
9. Корж А.А. «Масштабирование Data Intensive приложений с помощью библиотеки Dislib на суперкомпьютерах BlueGene/P и Ломоносов» // Материалы Всероссийской научной конференции «Научный сервис в сети ИНТЕРНЕТ», Новороссийск, 19-24 сентября 2011 г., Изд-во Московского Университета, с.126-131.
10. Murphy R., Kogge P. «On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and Its Implications» // IEEE Transactions on Computers 56(7), July 2007, 937–945.

11. Seshadhri C., Pinar A., Kolda T. «An In-Depth Study of Stochastic Kronecker Graphs» // Proceedings of the 2011 IEEE 11th International Conference on Data Mining (ICDM), 587–596.
12. Davis T., Hu Y. «The University of Florida Sparse Matrix Collection» // ACM Transactions on Mathematical Software, Vol. V, No. N, M 20YY, 1–28.
13. 10th DIMACS Implementation Challenge // URL:http://www.cc.gatech.edu/dimacs10/index.shtml (дата посещения: 01.06.2011).