АЛГОРИТМЫ СЕМЕЙСТВА DIAMONDTILE ДЛЯ ЭФФЕКТИВНОГО ВОЛНОВОГО МОДЕЛИРОВАНИЯ НА GPGPU

В.Д. Левченко

Институт прикладной математики им. М.В. Келдыша РАН

Введение Развитие высокопроизводительных вычислений последнего десятилетия тесно связано с адаптацией специализированных графических ускорителей к требованиям универсальных вычислений. В результате появились как новый класс устройств, использующих графические процессоры общего назначения (GPGPU), пиковая производительность которых примерно на порядок выше традиционных процессоров общего назначения (CPU), так и новая программная технология (CUDA), обеспечивающая удобство использования высокопараллельных вычислений разных уровней.

Развитие графических ускорителей шло по пути повышения пиковой вычислительной производительности, которая сейчас для одного процессора составляют несколько триллионов операций с плавающей точкой в секунду (Тфлопс). Так, выбранные ниже для целей тестирования видеокарты производства компании nVidia, Titan (процессор GK110, архитектура Kepler) и GeForce 750Ti (процессор GM107, архитектура Maxwell) имеют пиковую производительность на данных одинарной точности 4.7 Тфлопс и 1.4 Тфлопс соответственно, занимая позиции на противоположных концах ценового диапазона предложений современных игровых видеокарт. Вместе с тем, рост пропускной способности оперативной памяти видеокарт отставал от роста вычислительной производительности. Для указанных карт это значение составляет 288.4 ГБ/сек и 86.4 ГБ/сек соответственно.

Производительность большого числа актуальных приложений на современных вычислительных системах ограничиваются не пиковой производительностью, а пропускной способностью оперативной памяти. Характерным примером такой ситуации может быть и моделирование различных физических сред, в актуальных задачах которого размер данных, пересчитываемых на каждом шаге по времени, составляет от десятков гигабайт до терабайт. Для задач волнового моделирования стандартные алгоритмы предполагают декомпозицию области моделирования на небольшие подобласти, так что данные каждой могут храниться на одной видеокарте. При этом, после расчета одного шага по времени, требуется синхронизация с соседними подобластями с пересылкой данных граничных ячеек. Такой алгоритм неизбежно становится ограниченным в производительности пропускной способностью памяти или шин коммуникаций и следовательно, не использует в полной мере возможности возросшей вычислительной производительности.

Цель данной работы - построение для тех же задач и численных схем сеточных методов – семейства алгоритмов, допускающих, при выполнении определенных условий, достижения высокой вычислительной эффективности (в данной статье - отношения достигнутой производительности к пиковой).

В работе описываются локально-рекурсивные нелокально-асинхронные (LRnLA) алгоритмы класса DiamondTile, позволяющие с высокой эффективностью использовать новые возможности графических ускорителей в численном моделировании на основе сеточных методов. Разработанные алгоритмы находят приложения в актуальных задачах нанооптики, спинтроники, сейсморазведки, физики плазмы, газовой динамики.

Постановка задачи В качестве простого, но сохраняющего основные особенности примера численной схемы рассмотрим обычную для таких задач конечно-разностную явную схему второго порядка аппроксимации по координатам и времени для скалярного волнового уравнения

$$\frac{F^{k+1}_{i^x,i^y,i^z}-2F^{k}_{i^x,i^y,i^z}+F^{k-1}_{i^x,i^y,i^z}}{\Delta_t^2} =$$

$$= c^2 \left\{ \frac{F^{k}_{i^x+1,i^y,i^z}-2F^{k}_{i^x,i^y,i^z}+F^{k}_{i^x-1,i^y,i^z}}{\Delta_x^2} + \frac{F^{k}_{i^x,i^y+1,i^z}-2F^{k}_{i^x,i^y,i^z}+F^{k}_{i^x,i^y-1,i^z}}{\Delta_y^2} + \frac{F^{k}_{i^x,i^y,i^z+1}-2F^{k}_{i^x,i^y,i^z}+F^{k}_{i^x,i^y,i^z-1}}{\Delta_z^2} \right\}$$

в простой области трехмерного пространства (прямоугольном параллелепипеде со сторонами, совпадающими с декартовыми осями координат). Без ограничения общности, будем рассматривать дважды периодические граничные условия (по y и z), а по оси x - источники сигнала и/или отражение. После приведения подобных членов в вышеприведенном уравнении получаем явную схему обновления значения поля F в ячейке (ix,iy,iz)

$$F^{k+1}_{i^x,i^y,i^z} = k_0 F^{k}_{i^x,i^y,i^z} - F^{k-1}_{i^x,i^y,i^z} + k_x \left(F^{k}_{i^x+1,i^y,i^z}+F^{k}_{i^x-1,i^y,i^z}\right) + k_y \left(F^{k}_{i^x,i^y+1,i^z}+F^{k}_{i^x,i^y-1,i^z}\right) + k_z \left(F^{k}_{i^x,i^y,i^z+1}+F^{k}_{i^x,i^y,i^z-1}\right)$$

$$k_x = \frac{c^2\Delta_t^2}{\Delta_x^2}, \quad k_y = \frac{c^2\Delta_t^2}{\Delta_y^2}, \quad k_z = \frac{c^2\Delta_t^2}{\Delta_z^2}, \quad k_0 = 2\left(1 - k_x - k_y - k_z\right).$$

Таким образом, для пересчета значения поля F в одной ячейке на одном шаге по времени требуется 7 операций сложения и 4 операции умножения, или, учитывая особенности строения блоков вычислений современных процессоров - 7 операций FMA. Также, считая коэффициенты k постоянными, требуется загрузка восьми значений поля из соседних ячеек согласно шаблону «крест» (рис. 1) и сохранения одного результата.

При размере данных 4 байта (тип float) отношение требуемого обмена с памятью к числу операций составляет более 5 байт на FMA операцию. Однако для указанных выше карт искомый параметр составляет лишь 0.12 байт на FMA операцию, что в 40 раз меньше требуемого.

Особенности подсистемы памяти графических ускорителей Подобная проблема дисбаланса пропускной способности памяти относительно пиковой производительности имеет долгую историю и для традиционных компьютерных архитектур, что привело к развитию иерархии подсистемы памяти в виде многоуровневой пирамиды (рис.2, в центре). Наличие данной иерархии позволяет для указанных задач использовать алгоритмы, локализующие данные на верхних уровнях иерархии и снижающие требования к пропускной способности памяти во много раз. Так, алгоритмы LRnLA [1], последовательно реализуя такой подход, позволяют в реальных задачах достичь более трети от пиковой производительности многоядерных многопроцессорных систем с архитектурой NUMA.

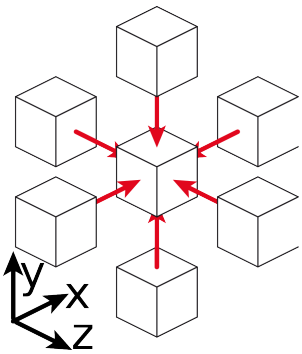

Рис. 1. Шаблон типа «крест»

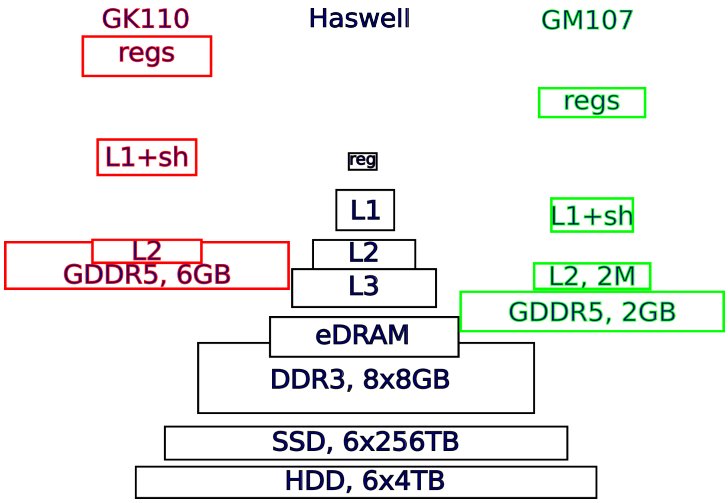

Рис. 2. Иерархия подсистемы памяти для GPGPU и CPU. На диаграмме ширина каждого прямоугольника соответствует логарифму от размера соответствующего уровня иерархии памяти, а высота его верхней грани - логарифму суммарной пропускной способности

Что касается подсистемы памяти графических ускорителей, то, как видно из того же рисунка, на данный момент ее иерархия содержит лишь два значимых по размеру уровня: нижним является память GDDR5 с пропускной способностью, соизмеримой с пропускной способности кэша L2/L3 современных CPU, а верхним - регистровый файл, суммарный размер которого соизмерим с размером кэша L3/L2, а суммарная пропускная способность на порядки выше пропускной способности регистрового файла CPU.

Таким образом, для снижения требований к пропускной способности памяти, наиболее эффективно локализовать данные в регистровом файле GPGPU. Подобная локализация используется, например, при блочном перемножении плотных матриц в библиотеке cublas [2], но в традиционных реализациях конечно-разностных методов для CUDA [3] данные локализуются в shared-памяти.

Для количественной характеристики степени локализации данных введем параметр локализации как отношение количества обновляемых данных к количеству загружаемых/сохраняемых данных. Для исходного «наивного» алгоритма, требующего при обновлении одной ячейки восьми загрузок и одного сохранения, это отношение составляет 1/9. Для наилучших алгоритмов с пошаговой синхронизацией, использующих перекрытия шаблонов при помощи тайлов (компактных блоков ячеек, данные которых заведомо помещаются в верхних уровнях иерархии памяти) - не выше 1/3, так как для обновления одного тайла требуется загрузка тайлов двух слоев по времени и сохранение одного тайла результата вычислений.

В итоге, на современных графических ускорителях производительность подобной реализации ограничена пропускной способности памяти, а достигаемая производительность не превышает значений 5-10 миллиардов ячеек в секунду (точки слева внизу на рис. 3), что составляет менее 5% от вычислительной эффективности. Для потенциальной возможности достижения пиковой производительности требуется увеличить параметр локализации до значений 4-5, что невозможно без многократного пересчета данных ячеек, локализованных в регистровом файле вместе со своими соседями, входящими в шаблон численной схемы. Это в свою очередь требует замены пошаговой синхронизации отслеживанием зависимостей по данным для многих шагов по времени, и также замены тайлового разбиения области на декомпозицию графа зависимостей.

Построение оптимального алгоритма для шаблона «крест» Следуя общей методологии построения LRnLA алгоритмов, вначале требуется выбрать разбиение пространства данных (в данном случае прямоугольной сетки), на котором затем построить конусоиды зависимости-влияния, тем самым проведя разбиение графа зависимостей, и наконец, определить правила обхода полученных конусоидов. В работе [1] использовалось разбиение d-мерного пространства на d-мерные кубы, которое порождается шаблоном типа «квадрат» и оптимально при наличии в аппроксимируемом дифференциальном уравнении смешанных частных производных, а также в кинетических методах (решеточных уравнений Больцмана, частиц-в ячейках).

Шаблон типа «крест» является частным случаем шаблона типа «квадрат», но более локален (содержит меньшее число зависимостей) и в силу широкого распространения в методах конечных разностей, объемов и т.д., требует отдельного рассмотрения.

Формально два вышеперечисленных случая можно разделить введением расстояния Минковского в пространстве данных, с параметром p, определяемым шаблоном схемы. В первом случае получим метрику Чебышева (с бесконечным p), а во втором - метрику «городских кварталов» (p=1). Если в первом случае множество равноудаленных точек от центра ляжет на поверхность d-мерного куба, то во втором - на поверхность d-мерного октаэдра.

В частном случае двух и трех измерений, такую форму (ромб и октаэдр) обычно связывают с формой природного кристалла алмаза. По этой причине, описываемый класс алгоритмов будет иметь в своем названии Diamond. Вторая часть названия DiamondTile определяется аналогией с тайловыми алгоритмами. Отличие состоит в том, что тайлы заполняют модельную область (т. е. область трехмерного пространства), а DiamondTile – четырехмерное пространство операций (или граф зависимостей).

В этой работе ограничимся случаем d=2, что дает простое разбиение плоскости на ромбы. Поскольку моделирование происходит в трехмерной области, то, без ограничения общности выберем для этого плоскость xy. По оси z (обычно, с минимальным числом ячеек) используем векторизацию, сопоставив номер ячейки iz с номером CUDA-нити. Тем самым автоматически решаются проблемы выровненного векторизованного обмена с памятью видеокарты, а эффективность реализации алгоритма относительно утилизации доступной пропускной способности памяти находится в диапазоне 70-90%% (на рис.3 справа).

Вернемся к разбиению прямоугольной сетки в плоскости xy на ромбы. Элементарным (размером 1) алгоритмом DiamondTile с индексами (ix,iy), k (ix+iy и k - четные) назовем вычисления полей F двух пар ячеек, соседних по оси x и по времени t, имеющих координаты по x,y, соответственно (ix,iy) и (ix+1,iy) для шага по времени k и (ix+1,iy) и (ix+2,iy) для шага k+1. Заметим, что на текущем шаге k по времени DiamondTile с начальной координатой (ix,iy) непосредственно зависит от результатов выполнения трех других DiamondTile того же шага k, с индексами (ix+1,iy-1), (ix+1,iy+1) и (ix+2,iy).

Учитывая направление и характер этих зависимостей, нетрудно предложить оптимальный алгоритм заполнения всей области элементарными DiamondTile с синхронизацией через пару шагов по времени: двигаясь справа налево по оси x по одной ячейке, выполнять асинхронно DiamondTile для половины ячеек по y, четных при четном ix и нечетных при нечетном. Асинхронность выполнения позволяет оптимально задействовать CUDA-блоки.

Посчитаем параметр локализации для одного элементарного DiamondTile. Для обновления четырех значений полей (двух на шаге k+1 и двух на шаге k+2) требуется загрузить 16 значений полей по шаблону (2 значения с шага k-1, 8 значений с шага k и 6 значений с шага k+1), и затем сохранить 4 результата вычислений. В результате параметр локализации будет равен 1/5. Для соседних DiamondTile, выполняемых параллельно в CUDA-блоках, он будет чуть выше (до 1/4) за счет оседания данных перекрывающихся шаблонов в кэше L2 (например, для асинхронных DiamondTile с индексами (ix,iy),k и (ix,iy+2),k в их общий шаблон входят четыре значения поля в ячейках (ix,iy+1), (ix+1,iy+1) на шаге k и (ix+1,iy+1), (ix+2,iy+1) на шаге k+1).

Для дальнейшего повышения параметра локализации, элементарные DiamondTile можно объединять в более крупные DiamondTile (с линейным размером n). Для этого вначале для момента времени k вычислим поля в основании в виде ромба, состоящим из n*n пар ячеек, а затем будем сдвигать этот ромб 2*n-1 раз на одну ячейку по x, вычисляя поля для следующего шага по времени. При этом для крупных DiamondTile зависимости и правила обхода остаются теми же, что и для элементарного, с точностью до замены индексов (ix,iy), k на (n*ix,n*iy), n*k. Значение параметра локализации при этом будет равно n/(6-3/n+1/n/n), которое можно еще повысить до n/(4-1/n), объединив несколько DiamondTile в достаточно высокую «башню» DiamondTorre таким образом, чтобы нижнее основание каждого следующего DiamondTile совпадало с верхним основанием предыдущего.

В итоге, параметр локализации превышает единицу уже при n=4, двойку при n=8 и пятерку при n=20. Последнее значение позволит достичь пиковой производительности в рассмотренной задаче на современных графических ускорителях (хотя бы потенциально).

Результаты тестирования В текущей реализации описанного алгоритма имеющегося размера регистрового файла хватает для локализации данных DiamondTorre с n<=7, что дает максимальный параметр

локализации 1.8 и потенциальную возможность превысить треть от пиковой производительности. Результаты тестирования (рис.3) показывают обоснованность сделанных предположений.

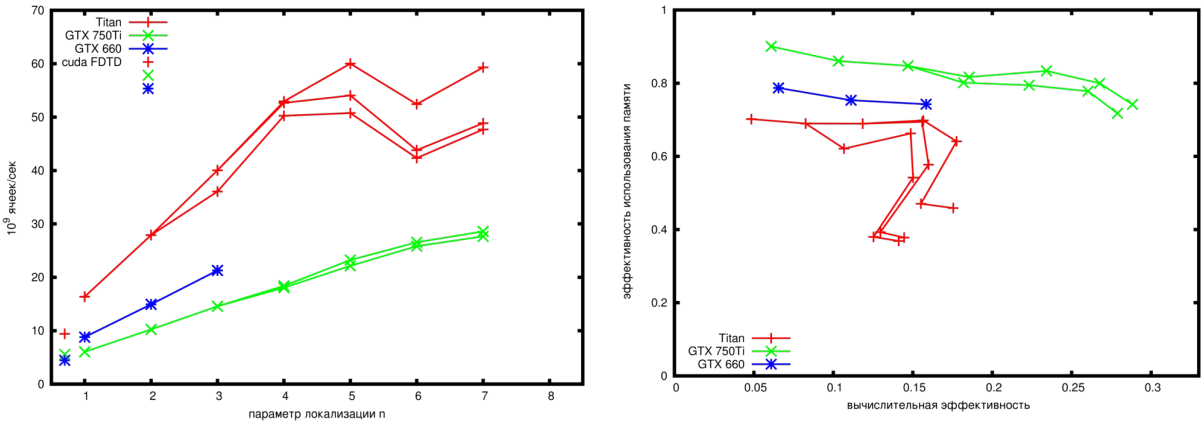

Рис. 3. Зависимость производительности при решении волнового уравнения алгоритмом DiamondTile от n, определяющего значение параметра локализации (слева), те же данные, отнормированные на пиковую производительность по оси абсцисс и на пиковую пропускную способность памяти по оси ординат (справа). На графиках приведено три набора данных, соответствующих разной параллельной загруженности мультипроцессоров

Так, для всех протестированных видеоускорителей производительность растет почти линейно с ростом n и пропорциональным ему параметром локализации, а для процессора новой архитектуры Maxwell (GM107) вычислительная эффективность достигает 30%. Насыщение производительности и падение эффективности при больших n для процессора предыдущей архитектуры Kepler (GK110) связано не только с малым размером кэша второго уровня в архитектуре Kepler (меньшим, чем размер локализуемых в регистровом файле данных), но и с сопутствующим росту n снижением параллельной загруженности («occupancy») из-за увеличенных требований к количеству регистров. В результате, для архитектуры Kepler, содержащей 192 FMA блоков в одном мультипроцессоре (общий регистровый файл размером 256Кбайт) против 128 в архитектуре Maxwell, именно недостаточная параллельная загруженность становится узким местом.

Для проверки этого тезиса было проведено варьирование числа ячеек Nz по оси z, что в данной реализации определяет количество CUDA-нитей и CUDA-блоков, обрабатываемых одним мультипроцессором, что в свою очередь определяет параллельную загруженность. Результаты также представлены на рис.3 в виде трёх наборов данных для каждого видеоускорителя (с Nz, оптимальным с точки зрения параллельной загруженности, с Nz не меньшим 64, и с Nz не меньшим половины максимально возможного, от 1024 до 256 в зависимости от n). Видно, что результаты существенно различаются лишь для больших n и для архитектуры Kepler. Подобная проблема наблюдалась также в ранних версиях библиотеки cublas, и позднее была преодолена с использованием недокументированных оптимизационных приёмов (включая ассемблерный код). В рамках данной работы мы ограничились обычными приёмами высокоуровневого программирования и лишь алгоритмическими оптимизациями. Отдельно следует отметить, что по сравнению с традиционной тайловой оптимизацией, реализация, основанная на использовании алгоритмов DiamondTile достигает в 6 раз большей производительности.

Заключение Разработаны новые алгоритмы DiamondTile семейства LRnLA, эффективные для модели памяти графических ускорителей, для численного решения широкого класса задач моделирования, дискретизация которых приводит к шаблонам зависимости типа «крест».

Программная реализация решения волнового уравнения достигает производительности 50-60 млрд. ячеек/сек на GK110 (10-20%% вычислительной эффективности) и 27-29 млрд. ячеек/сек на GM107 (30% эффективности), при этом «узким местом» вместо ограничения пропускной способности памяти становится недостаточная параллельная загруженность.

Работа поддержана грантом РФФИ 12-01-00708-а

ЛИТЕРАТУРА:

1. В.Д. Левченко Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: все грани параллелизма», г. Новороссийск, 23-28 сентября 2013 г, М.: Издательство МГУ им. М.В.Ломоносова, 2013. С.98–103.
2. V. Volkov, J.W. Demmel Benchmarking GPUs to tune dense linear algebra, Proceedings of the 2008 ACM/IEEE conference on Supercomputing, November 15-21, 2008, Austin, Texas
3. P. Micikevicius 3D finite difference computation on GPUs using CUDA, Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, p.79-84, March 08-08, 2009, Washington, D.C