Использование технологий CUDA и OpenCL для моделирования сейсмических процессов сеточно-характеристическим методом*

А.М. Иванов[1], Н.И. Хохлов[1]

Московский физико-технический институт (государственный университет)[1]

В работе рассматриваются технологии CUDA и OpenCL применительно к задачам сейсмики в упругих средах. Решается задача распространения динамических волновых возмущений в геологической среде в упругой постановке в двумерном случае. Для численного решения используется сеточно-характеристический метод. Производительность алгоритма решения задачи на GPU сравнивается с производительностью решения на одном ядре центрального процессора. Исследуется влияние различных оптимизаций на производительность алгоритма. Измеряется эффективность распараллеливания на нескольких графических процессорах.

Ключевые слова: сейсмика, сеточно-характеристический метод, CUDA, OpenCL.

1. Введение

В последнее время все более интенсивно используются технологии высокопроизводительных вычислений на графических процессорах. Эти технологии хорошо применимы к задачам сейсмики в упругих средах, так как они требуют большого количества вычислительных ресурсов. Для решения этих задач требуется численное решение гиперболических систем уравнений.

В некоторых работах производится решение различных задач сейсмики, которые сводятся к решению гиперболических систем уравнений, на GPU. В работе [1] описывается реализация численного метода ADER-DG с использованием технологии CUDA. В других работах изучается реализация WENO схем на GPU [2]. Исследуется ускорение при решении гиперболических систем уравнений на структурированных сетках на GPU по сравнению с CPU [3].

К решению гиперболических систем уравнений сводятся и другие задачи. В работе [4] производятся вычисления на GPU с повышенной точностью - до 60 знаков после запятой. Авторы [8, 9] получили прирост производительности при решении уравнений мелкой воды. Моделируются магнитогидродинамические процессы на графических процессорах [5]. Авторами [10] на графических процессорах реализуется разрывный метод Галеркина и детально описывается его профилирование. Тот же метод был реализован на нескольких GPU в работе [11] при этом получено ускорение в 28.3 раза на GPU кластере по сравнению с CPU кластером. В работе [12] также сравнивается ускорение на кластере GPU по сравнению с CPU кластером, в вычислениях методом спектральных элементов распространения сейсмических волн.

Помимо этого технология CUDA применяется для решения других задач. Показывается влияние большого количества операций ввода-вывода на производительность алгоритма [13]. Сравнивается производительность GPU и CPU [14–17]. При этом делаются попытки снизить энергопотребление графическими процессорами во время вычислений [18]

В данной работе используется сеточно-характеристический метод, который хорошо зарекомендовал себя для задач сейсмики [20], в которой требуется численное решение гиперболических систем уравнений [19]. Данный метод хорошо поддается распараллеливанию, поскольку используется явный метод и большие расчетные сетки.

*Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта 16-29-15097 офи_м.

Ранее алгоритм уже был распараллелен используя технологии MPI и OpenMP. В данной работе алгоритм был реализован с использованием технологий CUDA и OpenCL. Было рассмотрено влияние различных оптимизаций на производительность реализации алгоритма с использованием технологии CUDA. После этого наиболее эффективная реализация была переписана с использованием технологии OpenCL. Кроме того, алгоритм задействует несколько графических процессоров, поэтому была измерена эффективность распараллеливания на нескольких GPU. Для тестирования были использованы карты от NVIDIA и AMD. На GPU NVIDIA результаты работы OpenCL реализации сравнивались с такой же реализацией на CUDA. От NVIDIA использовались карты GeForce и Tesla, в том числе и последние модели Tesla k80 и Tesla k40m. От AMD использовались GPU серий Radeon HD и Radeon R9. Были рассмотрены различия в эффективности реализации с одинарной точностью и с двойной точностью. В качестве центрального процессора для тестирования последовательной реализации использовался CPU Intel Xeon E5-2697.

2. Математическая модель

Поведение среды описывается моделью идеального изотропного линейно-упругого материала. Рассматривается двумерная задача. Следующая система дифференциальных уравнений в частных производных описывает состояние элементарного объёма упругого материала в приближении малых деформаций:

$$\rho\frac{\partial v_x}{\partial t} = \frac{\partial \sigma_{xx}}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y}, \rho\frac{\partial v_y}{\partial t} = \frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_{yy}}{\partial y},$$
$$\frac{\partial \sigma_{xx}}{\partial t} = (\lambda+2\mu)\frac{\partial v_x}{\partial x} + \lambda\frac{\partial v_y}{\partial y}, \frac{\partial \sigma_{yy}}{\partial t} = \lambda\frac{\partial v_x}{\partial x} + (\lambda+2\mu)\frac{\partial v_y}{\partial y}, \frac{\partial \sigma_{xy}}{\partial t} = \mu\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y}\right),$$

где ρ — плотность среды; λ, μ — параметры Ламе; v_x и v_y — горизонтальная и вертикальная составляющие скорости частиц среды; $\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$ — компоненты тензора напряжения.

Данную систему можно переписать в матричной форме:

$$\frac{\partial \mathbf{u}_p}{\partial t} + A_{pq}\frac{\partial u_q}{\partial x} + B_{pq}\frac{\partial u_q}{\partial y} = 0, \tag{1}$$

где $\mathbf{u}$ — вектор из 5 независимых переменных $\mathbf{u} = (\sigma_{xx}, \sigma_{yy}, \sigma_{xy}, v_x, v_y)^T$. Явный вид матриц A_{pq}, B_{pq} представлен в [6]. Здесь и далее подразумевается суммирование по повторяющимся индексам. Собственные значения матриц A_{pq} и B_{pq} таковы: $s_1 = -c_p, s_2 = -c_s, s_3 = 0, s_4 = c_s, s_5 = c_p$, где c_p и c_s — скорости распространения продольных и поперечных волн в среде.

3. Численный метод

Применяя покоординатное расщепление, можно свести задачу построения разностной схемы для системы уравнений (1), к задаче построения разностной схемы для системы вида:

$$\frac{\partial u_p}{\partial t} + A_{pq}\frac{\partial u_q}{\partial x} = 0 \tag{2}$$

Для гиперболической системы уравнений (2) матрицу $\mathbf{A}$ можно представить в виде $\mathbf{A} = \mathbf{R}\mathbf{\Lambda}\mathbf{R}^{-1}$, где $\mathbf{\Lambda}$ — диагональная матрица, элементы которой — собственные значения матрицы $\mathbf{A}$, а $\mathbf{R}$ — матрица, состоящая из правых собственных векторов матрицы $\mathbf{A}$. Введём новые переменные: $\mathbf{w} = \mathbf{R}^{-1}\mathbf{u}$ (так называемые инварианты Римана). Тогда система уравнений (2) сведётся к системе из 5 независимых скалярных уравнений переноса.

Приведем схему третьего порядка точности для численного решения одномерного линейного уравнения переноса $u_t + au_x = 0, a > 0, \sigma = a\tau/h$, τ — шаг по времени, h — шаг по

координате:

$$
\begin{aligned}
&u_m^{n+1} = u_m^n + \sigma(\Delta_0 + \Delta_2)/2 + \sigma^2(\Delta_0 - \Delta_2)/2 + \frac{\sigma(\sigma^2-1)}{6}(\Delta_1 - 2\Delta_0 + \Delta_2),\\
&\Delta_0 = u_{m-1}^n - u_m^n,\\
&\Delta_1 = u_{m-2}^n - u_{m-1}^n,\\
&\Delta_2 = u_m^n - u_{m+1}^n.
\end{aligned}
\tag{3}
$$

Схема (3) устойчива для чисел Куранта не превышающих единицу. Используется сеточно-характеристический критерий монотонности, опирающийся на характеристическое свойство точного решения:

$$min(u_{m-1}^n, u_m^n) \le u_m^{n+1} \le max(u_{m-1}^n, u_m^n).$$

В местах выполнения данного критерия порядок схемы падает до второго.

После того как значения инвариантов Римана на следующем шаге по времени найдены, восстанавливается решение: $\mathbf{u}^{n+1} = \mathbf{R}\mathbf{w}^{n+1}$.

4. Постановка задачи

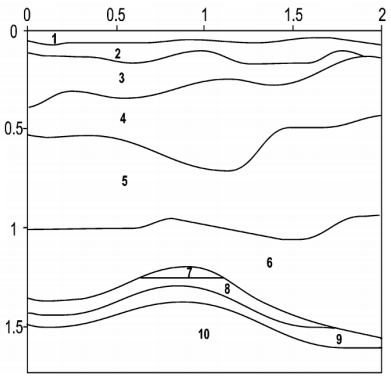

Рис. 1. Геологическая модель антиклинальной ловушки [7]

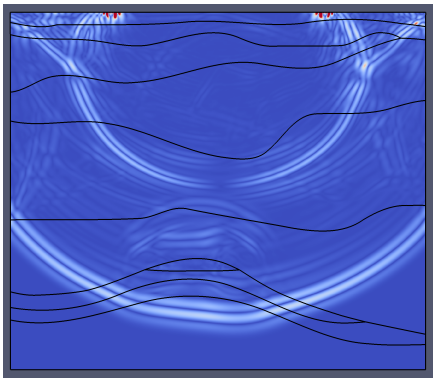

Рис. 2. Результат расчета – волновая картина в момент времени $t = 0.38$ с

Тестовая модель изображена на рис. 1. Размеры области даны в километрах. На нижней и боковых границах устанавливалось неотражающее граничное условие, на верхней – свободная граница. В качестве источника возмущения использовалась вертикальная сила, приложенная к площадке с 925.7 м по 974.1 м на дневной поверхности, амплитуда которой описывалась импульсом Рикера частоты 40 Гц. Результаты расчета представлены на рис. 2.

5. Условия тестирования

Рассматривалась двумерная тестовая задача с количеством узлов 4096×4096. Осуществлялось 6500 шагов по времени. В каждом узле сетки хранилось по 5 переменных с плавающей точкой. Все вычисления производились как с одинарной (SP), так и с двойной (DP) точностью. Размер сетки в памяти — 320 Мбайт для вычислений с одинарной точностью и 640 Мбайт для вычислений с двойной точностью. Однако различные оптимизации требовали большего количества памяти.

Если количество потоковых процессоров (ядер CUDA) – C, частота – F, то количество GFLOPS для одинарной точности – $2CF$, где 2 стоит по причине того, что за такт может выполняться 2 операции FMA(fused multiply-add). Для различных архитектур процессоров известно сколько в них содержится блоков(unit'ов) одинарной точности и двойной точности.

Таблица 1. Характеристики тестируемых графических карт

Графический процессор	Потоковые процессоры	Тактовая частота, МГц	ГФлопс (SP)	SP:DP	ГФлопс (DP)
GeForce GT 640	384	900	691	24	29
GeForce GTX 480	480	1401	1345	8	168
GeForce GTX 680	1536	1006	3090	24	129
GeForce GTX 760	1152	980	2258	24	94
GeForce GTX 780	2304	863	3977	24	166
GeForce GTX 780 Ti	2880	876	5046	24	210
GeForce GTX 980	2048	1126	4612	32	144
Tesla M2070	448	1150	1030	2	515
Tesla K40m	2880	745	4291	3	1430
Tesla K80	2496	562	2806	1.5	1870
Radeon HD 7950	1792	800	2867	4	717
Radeon R9 290	2560	947	4849	8	606

В столбце SP:DP – их отношение. Из этого можно вычислить количество GFLOPS для двойной точности.

6. Описание алгоритма

За основу для реализации алгоритма на графических процессорах была взята оптимизированная для выполнения на центральном процессоре версия данной программы. Оптимизациям была подвергнута наиболее затратная в вычислительном плане часть алгоритма. Так как использовалось расщепление по пространственной координате, для пересчета всей сетки требовалось два шага: по оси X и по оси Y. При этом было вычислено количество арифметических операций с плавающей точкой, требуемое для пересчета одного узла сетки за два шага – 190 Флопс. Таким образом, зная количество узлов сетки, количество шагов по времени, можно было определить теоретически потребляемое алгоритмом количество ГФлопс. Далее, зная количество потоковых процессоров в графическом процессоре, их тактовую частоту и число FMA(fused multiply-add) процессоров в одном процессоре, была вычислена пиковая производительность для каждого GPU. Реальные тесты алгоритма на графических процессорах показали меньшие значения производительности. На рисунке 5 приведен результат тестирования для чисел с одинарной точностью, на рисунке 6 – с двойной. Процент от пиковой производительности алгоритмов измерялся как соотношение между этими двумя значениями — теоретически требуемым количеством Флопс для пересчета сетки и реально потребленным количеством Флопс.

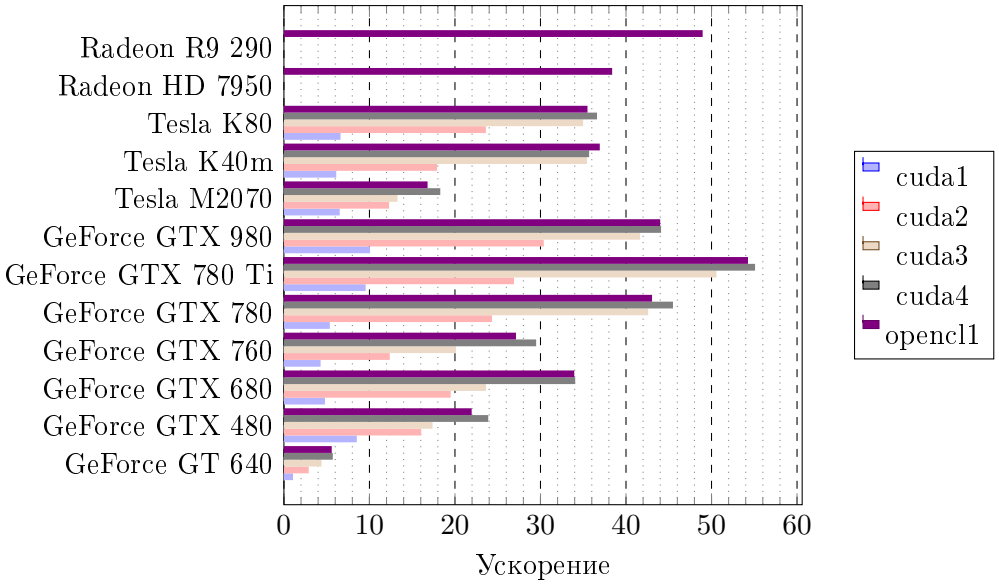

Рис. 3. Ускорение на GPU по сравнению с одним ядром CPU, одинарная точность

6.1. Перенос реализации с CPU на GPU - cuda1

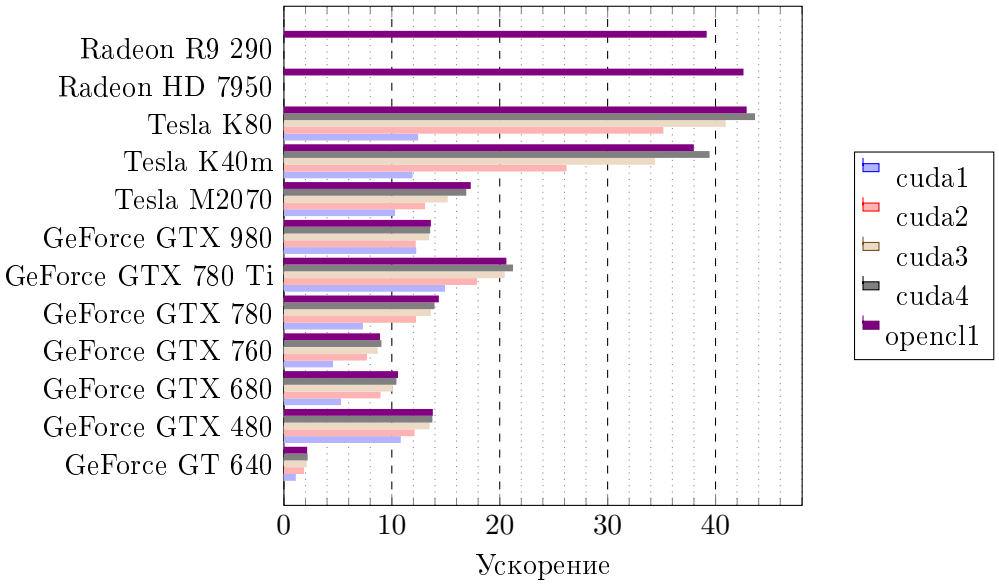

Рис. 4. Ускорение на GPU по сравнению с одним ядром CPU, двойная точность

В первоначальном варианте, алгоритм был перенесен на выполнение на графических процессорах с использованием технологии CUDA, но не оптимизирован для выполнения на GPU. В ней на графическом процессоре выделялось в 2 раза больше памяти, чем требовалось для хранения расчетной сетки. Это стандартная практика при работе с технологиями написания алгоритмов для графических процессоров. В результате происходит синхронизация только между вызовами функций, исполняющихся на графическом процессоре (CUDA kernels). Это было сделано по причине того, что глобальная синхронизация всего графический процессор создает большие временные задержки, поэтому эти задержки были вставлены между вызовами kernel'ей. Таким образом, удалось уменьшить количество глобальных синхронизаций до двух раз на один шаг по времени. Архитектура потоковых

процессоров CUDA такова, что в одном CUDA блоке потоки, исполняющие один и тот же код над разными областями памяти, выполняются одновременно, если нет ветвлений в коде программы. Поэтому ситуации, когда все потоки ждут завершения выполнения одного, случаются, только если этот один поток выполняет какие-то отличные от остальных операции.

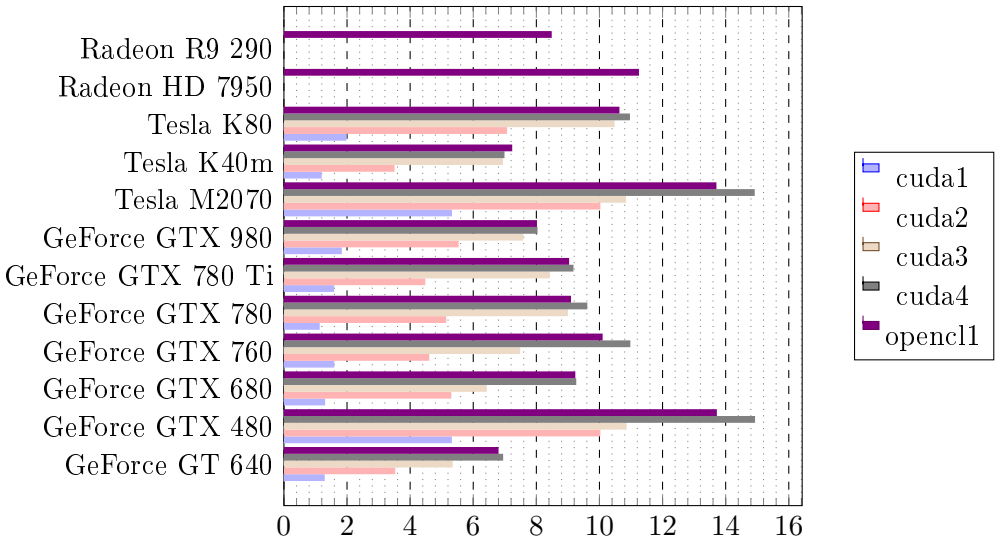

Рис. 5. Процент от пиковой производительности, одинарная точность

Все операции по выделению памяти на GPU и вызовам функций, исполняющихся на графическом процессоре, производится хостом – CPU. Операции копирования сетки из памяти хоста и обратно требуют много времени, поэтому сетка копируется один раз из памяти хоста в память графического процессора, перед началом основных вычислений, и один раз в конце из памяти GPU в память хоста. Количество шагов по времени и размер расчетной сетки были таковы, что время, требуемое для расчетов, намного превосходило время, требуемое на это копирование.

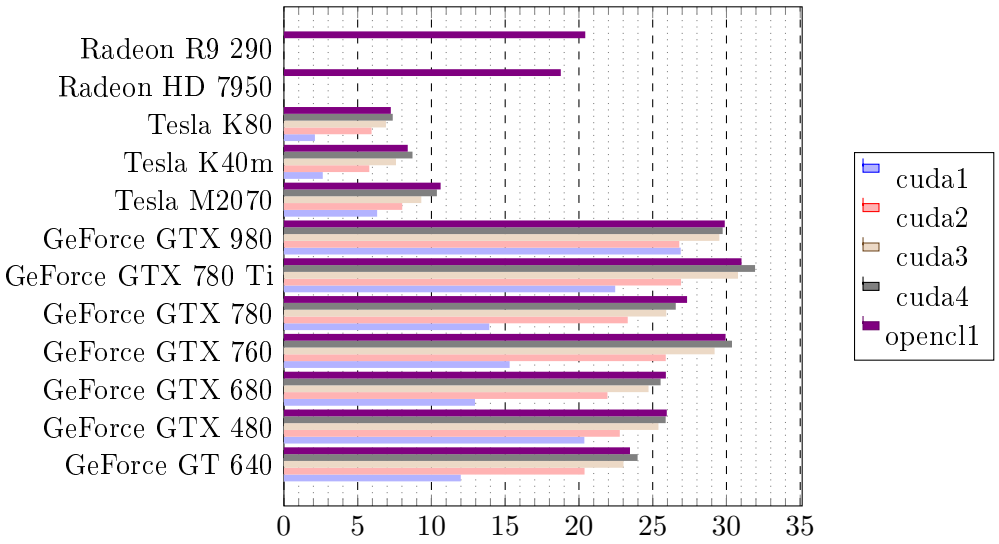

Рис. 6. Процент от пиковой производительности, двойная точность

Данные в сетке можно хранить двумя способами: в виде массива структур и в виде структуры массивов. Под структурой в данном случае подразумевается вектор u, состоящий из 5 компонент. В первоначальной версии, в расчетной сетке данные были организованы в

виде массива структур, то есть данные определенного узла лежат в памяти последовательно.

Для задания граничных условий задачи, требовалось отличное от остальных потоков поведение на границе расчетной сетки. Для обработки границ сетки, в коде kernel'а помещен участок кода с пятью блоками условными блоками if-else.

Для этой версии было перебрано несколько вариантов размера блока. Был найден оптимальный размер - 16x16, при котором время работы данной версии было минимально.

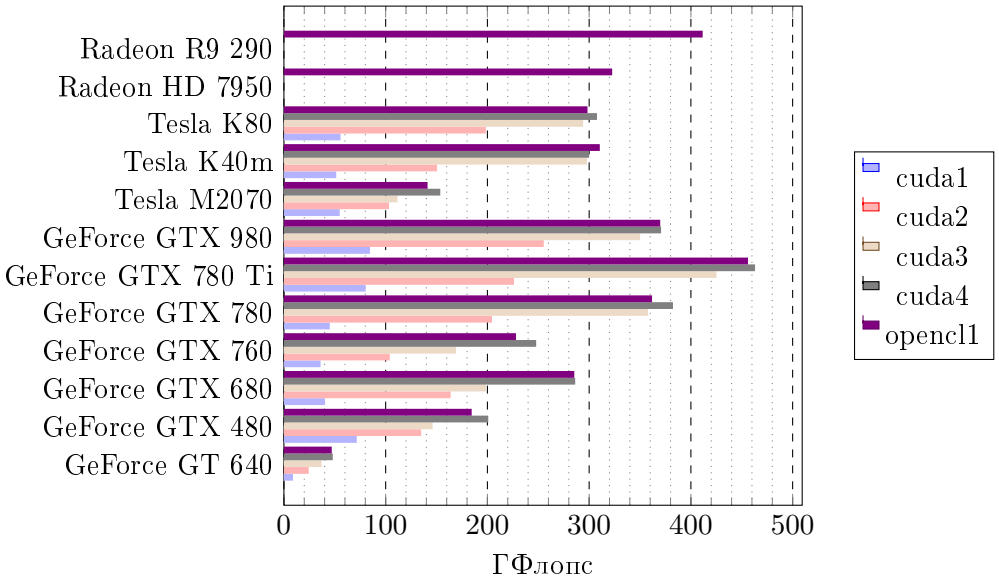

Рис. 7. Производительность, одинарная точность

6.2. Структура массивов и последовательное обращение к памяти - cuda2

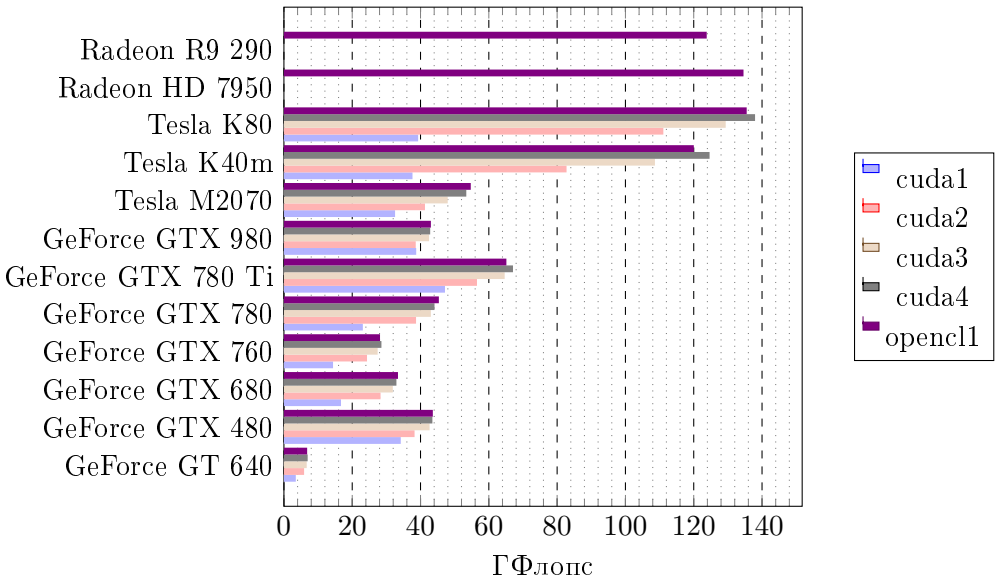

Рис. 8. Производительность, двойная точность

Следующая оптимизация состояла в использовании общей памяти (shared memory) блока CUDA. Это делается по причине того, что задержка (latency) при взаимодействии с этой памятью меньше чем при взаимодействии с глобальной памятью. Оптимизация заключает-

ся в том, что чтение из глобальной памяти происходит только один раз на каждом шаге по X и по Y в начале вызова функции, выполняющейся на GPU, при этом данные копируются в общую память. Сразу же после этого осуществляется синхронизация всех потоков в блоке, после чего все вычисления производятся над данными в общей памяти блока. В конце kernel'a результат записывается на вторую копию расчетной сетки в глобальной памяти.

Другая оптимизация заключалась в выборе другого способа хранения расчетной сетки в памяти графического процессора -– структуры массивов. После этого обращения к глобальной памяти стали последовательными (coalesced), что привело к повышению производительности.

Указанные выше оптимизации позволили уменьшить количество блоков if-else при обработке границ сетки до двух.

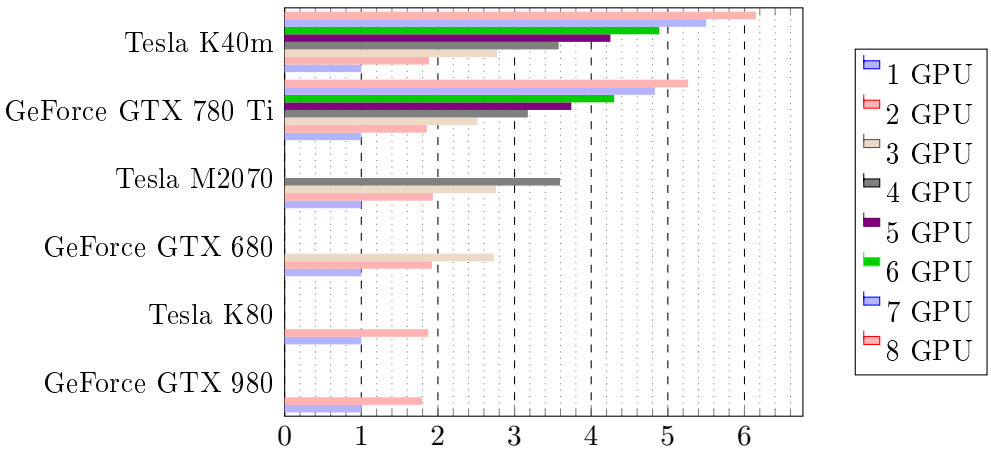

Рис. 9. Ускорение при синхронизации через память хоста, CUDA, одинарная точность

6.3. Параметры вызова kernel'a - cuda3

До этого дополнительные данные, такие как размер сетки, материал, значения переменных, полученных в результате промежуточных вычислений, справедливых на протяжении всех шагов по времени, передавались в качестве указателя на структуру в глобальной памяти графического процессора. Все потоки постоянно обращались к одному и тому же участку памяти. В версии алгоритма CUDA3, такие данные вычислялись на CPU один раз перед началом выполнения кода на GPU и передавались через параметры вызова kernel'a. Ожидалось, что у каждого потока будет создана локальная копия этих значений.

6.4. Размеры блоков - cuda4

Важно было подобрать размеры блоков так, чтобы графический процессор был постоянно загружено, то есть, чтобы не было частично не занятых потоковых процессоров. Также важно было выбрать размеры блоков кратные размерам warp'ов, так как внутри warp'a осуществляется последовательный доступ к памяти. Другая причина – потоки одного warp'a способны выполнять код одновременно над разными участками памяти, если нет ветвлений.

В версии CUDA4 подбираются размеры блоков таким образом, чтобы удовлетворить указанным выше требованием и минимизировать количество узлов, для которых требуются обмены памятью между блоками. В шаге по X для пересчета каждого узла расчетной сетки требуются значения в двух смежных узлах по оси X, а в шаге по оси Y – требуется два смежных узла по оси Y. Поэтому следовало подобрать размеры блоков так, чтобы количество узлов, для которых требуются значения в смежных блоках, было можно меньшим.

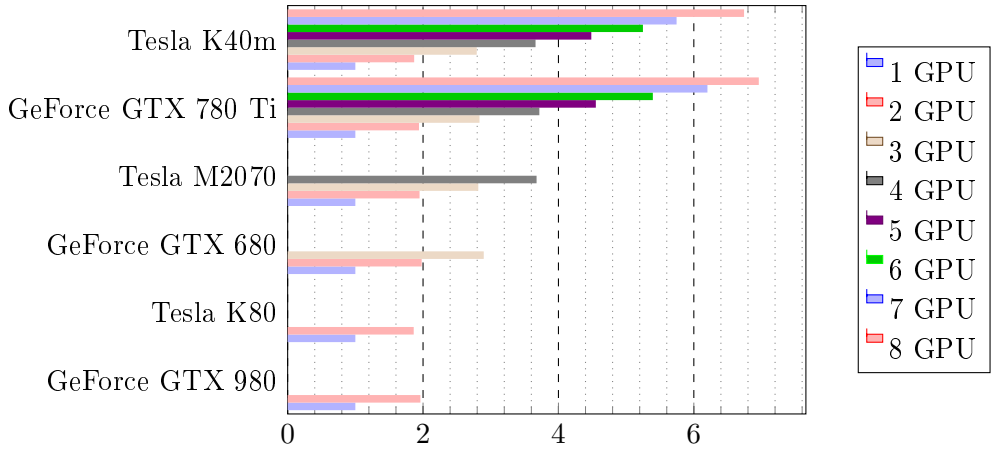

Рис. 10. Ускорение при синхронизации через память хоста, CUDA, двойная точность

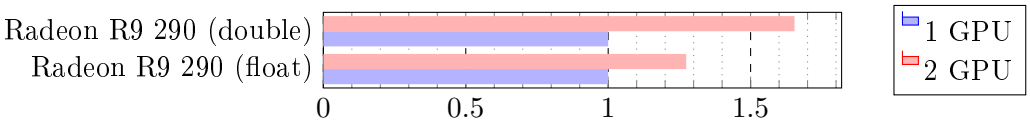

Рис. 11. Ускорение при синхронизации через память хоста, OpenCL

Это необходимо было сделать, так как для узлов смежных блоков требуется выделение дополнительной памяти в области общей памяти блока.

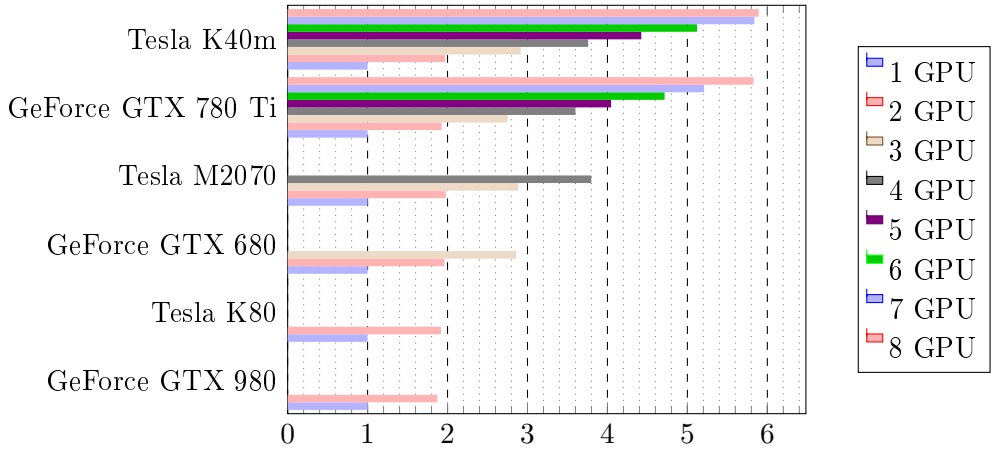

Рис. 12. Ускорение при использовании GPUDirect, одинарная точность

Если, например, взять блок размера $M \times N$, то в шаге по X для хранения узлов смежных блоков потребуется $4N$ дополнительных узлов, а в шаге по Y – $4M$. Поэтому в шаге по X размер был выбран равным 256×1, в результате блок требовал всего 4 дополнительных узла в памяти. В шаге по Y размер блока был принят равным 16×16, чтобы достичь компромисса между количеством дополнительной памяти (64 узла сетки) и требованиями к последовательному доступу к памяти.

6.5. opencl1

Следующим шагом было написание OpenCL реализации данного алгоритма. За основу для этого была взята оптимизированная версия алгоритма CUDA4.

Результаты тестирования ускорения для всех реализаций представленны на рисунках 3

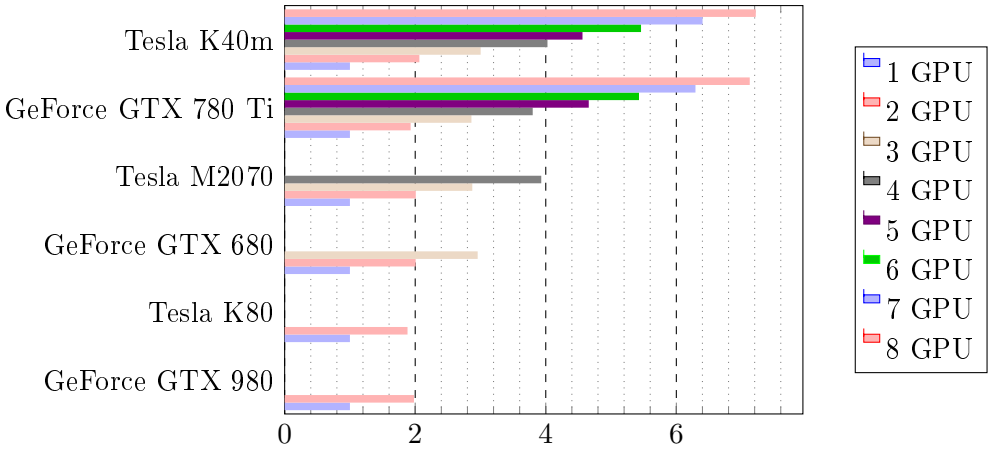

Рис. 13. Ускорение при использовании GPUDirect, двойная точность

и 4 для одинарной и двойной точности соответственно. Максимальное полученное ускорение по сравнению с CPU на одном графическом процессоре -– в 55 раз на GeForce GTX 780 Ti в вычислениях с одинарной точностью и в 44 раза на Tesla K80 в вычислениях с двойной точностью. Следует отметить хорошие результаты для устройств от AMD, несмотря на то, что использовались дешевые десктопные карты, они показали хороший результат на реализациях и одинарной и двойной точностью.

7. Несколько графических процессоров

Далее алгоритм был распараллелен для работы на нескольких графических процессорах. Для выполнения на нескольких GPU использовался наиболее оптимизированный вариант. Расчетная сетка при этом была разделена на несколько равных по размеру прямоугольных участков. Деление производилось по оси Y, вследствие выбранного размера блока по оси X.

Тесты с использованием нескольких GPU были сделаны только для одинаковых графических процессоров. Это сделано из-за того, что при использовании GPU с разной производительностью более быстрые процессоры окажутся причиной простоя более медленных графических процессоров, и эффект от их совместного использования может оказаться менее заметным. Синхронизация расчетной сетки между графическими процессорами производилась путем обмена через память хоста(CPU). Причем синхронизация производилась только один раз на каждом временном шаге перед шагом по оси Y. Если размер сетки равен $M \times N$, а число процессоров – D, то число требуемых для синхронизации узлов сетки – $4M(D-1)$. Результаты тестирования для нескольких устройств приведены на рисунках 9 и 10 для случаев одинарной и двойной тоности. Для процессоров от AMD использовалась технология OpenCL, результат приведен на 11.

Также была реализована версия на основе GPUDirect. Основное преимущество технологии GPUDirect при решении поставленной задачи заключается в возможности пересылки данных, располагающихся в памяти графических процессоров, напрямую без участия хоста через шину PCI Express, т.е. нет необходимости копировать данные вначале с графического процессора на хост, а потом с хоста на другой графический процессор. Результаты тестирования приведены на рисунках 12 и 13.

При использовании технологии CUDA результат отличается не значительно, при использовании в качестве буфера обмена память хоста. Для технологии OpenCL результат несколько хуже.

8. Выводы

В работе было показано, как были задействованы возможности графических процессоров, для решения задач сейсмики сеточно-характеристическим методом. Были описаны методы, которые позволили достигнуть наибольшей производительности указанного алгоритма при вычислениях на GPU. Рассмотрен вопрос эффективного использования памяти графического процессора как в случае использования одного GPU, так и в случае использования нескольких графических процессоров. Было определено влияние различных оптимизаций на производительность алгоритма. Максимальное полученное ускорение по сравнению с CPU на одном графическом процессоре -– в 55 раз на GeForce GTX 780 Ti в вычислениях с одинарной точностью и в 44 раза на Tesla K80 в вычислениях с двойной точностью. Для одинарной точности удалось достичь производительности в 460 ГФлопс, а для двойной точности была получена максимальная производительность в 138 ГФлопс. Максимальное достигнутое ускорение от количества графических процессоров – в 7.1 раз на 8 графических процессорах для двойной точности. Технология GPUDirect повысила ускорение до 10% от того, что было достигнуто без нее при вычислениях с одинарной точностью.

На основе полученных результатов можно сделать выводы о возможности променения графических процессоров для такого рода задач. Полученные результаты будут похожи для других задач, использующих похожие численные методы (метод конечных объемов, конечно-разностные методы). Также стоит отметить о хороших результатах для процессоров от производителя AMD и технологии OpenCL, в то время как большинство работ используют технологию CUDA для графических процессоров от NVidia.

Литература

1. Castro C.E., Behrens J.. Pelties C. CUDA-C implementation of the ADER-DG method for linear hyperbolic PDEs // Geoscientific Model Development Discussions. 2013. Vol. 80, No. 3. P. 3743–3786

2. Esfahanian V., Darian H.M., Gohari S.I. Assessment of WENO schemes for numerical simulation of some hyperbolic equations using GPU // Computers & Fluids. 2013. Vol. 6. P. 260–268

3. Rostrup S., De Sterck H. Parallel hyperbolic PDE simulation on clusters: Cell versus GPU // Computer Physics Communications. 2010. Vol. 181, No. 12. P. 2164–179

4. Khanna G. High-Precision Numerical Simulations on a CUDA GPU: Kerr Black Hole Tails // Journal of Scientific Computing. 2013. Vol. 56, No. 2. P. 366–380

5. Wong H.C., Wong U.H., Feng X., Tang Z. Efficient magnetohydrodynamic simulations on graphics processing units with CUDA // Computer Physics Communications. 2011. Vol. 182, No. 10. P. 2132–2160

6. LeVeque R. J. Finite volume methods for hyperbolic problems. Cambridge university press, 2002. 588 p.

7. Jose M. Review Article: Seismic modeling // Geophysics. 2002. Vol. 67, No. 4. P. 1304–1325.

8. Gallardo J.M., Ortega S., de la Asuncion M., Mantas J.M. Two-dimensional compact third-order polynomial reconstructions. solving nonconservative hyperbolic systems using gpus // Journal of Scientific Computing. 2011. Vol. 48, No. 1–3. P. 141–163.

9. Castro M. J., Ortega S., de la Asuncion M., Mantas J. M., Gallardo J. M. GPU computing for shallow water flow simulation based on finite volume schemes // Comptes Rendus Mecanique. 2011. Vol. 339, No. 2–3. P. 165–184.

10. Fuhry M., Giuliani A., Krivodonova L. Discontinuous galerkin methods on graphics processing units for nonlinear hyperbolic conservation laws // International Journal for Numerical Methods in Fluids. 2014. Vol. 76, No. 12. P. 982–1003.

11. Mu D., Chen P., Wang L. Accelerating the discontinuous galerkin method for seismic wave propagation simulations using multiple gpus with cuda and mpi // Earthquake Science. 2013. Vol. 26, No. 6. P. 377–393.

12. Komatitsch D. Fluid-solid coupling on a cluster of GPU graphics cards for seismic wave propagation // Comptes Rendus Mecanique. 2011. vol. 339, No. 2–3. P. 125–135.

13. Mielikainen J., Huang B., Huang H., Goldberg M. Improved gpu/cuda based parallel weather and research forecast (wrf) single moment 5-class (wsm5) cloud microphysics // Selected Topics in Applied Earth Observations and Remote Sensing. 2012. Vol. 5. P. 1256–1265.

14. Nickolls J., Buck I., Garland M., Skadron K. Scalable parallel programming with cuda // Queue. 2008. Vol. 6. P. 40–53.

15. Priimak D. Finite difference numerical method for the superlattice boltzmann transport equation and case comparison of cpu(c) and gpu(cuda) implementations // Journal of Computational Physics. 2014. Vol. 278. P. 182–192.

16. Tuttafesta M., Dangola A., Laricchiuta A., Minelli P., Capitelli M., Colonna G. GPU and multi-core based reaction ensemble monte carlo method for non- ideal thermodynamic systems // Computer Physics Communications. 2014. Vol. 185, No. 2. P. 540–549.

17. Ferroni F., Tarleton E., Fitzgerald S. GPU accelerated dislocation dynamics // Journal of Computational Physics. 2014. Vol. 272. P. 619–628.

18. Tamascelli D., Dambrosio F. S., Conte R., Ceotto M. GPU Accelerated Semiclassical Initial Value Representation Molecular Dynamics // ArXiv e-prints. 2013.

19. Karavaev D. A., Glinsky B. M., Kovalevsky V. V. A technology of 3D elastic wave propagation simulation using hybrid supercomputers // СУПЕРКОМПЬЮТЕРНЫЕ ДНИ В РОССИИ. Труды международной конференции. 2015. С. 26–33.

20. Хохлов Н.И., Петров И.Б. Применение современных технологий для высокопроизводительных вычислительных систем для решения задач локальной и глобальной сейсмики // СУПЕРКОМПЬЮТЕРНЫЕ ДНИ В РОССИИ. Труды международной конференции. 2015. С. 380–391.

Using CUDA and OpenCL technologies for modeling seismic processes with grid-characteristic method

A.M. Ivanov[1], N.I. Khokhlov[1]

Moscow Institute of Physics and Technology (State University)[1]

CUDA and OpenCL technologies applied to seismic problems in elastic media are considered in this paper. The problem is a two-dimensional elastic wave propagation. For numerical solution grid-characteristic method is used. Performance of algorithm on the GPU is compared with performance on a single CPU core. The influence of various optimizations on the performance of the algorithm is investigated.

Keywords: seismic, grid-characteristic, CUDA, OpenCL.

References

1. Castro C.E., Behrens J.. Pelties C. CUDA-C implementation of the ADER-DG method for linear hyperbolic PDEs // Geoscientific Model Development Discussions. 2013. Vol. 80, No. 3. P. 3743–3786

2. Esfahanian V., Darian H.M., Gohari S.I. Assessment of WENO schemes for numerical simulation of some hyperbolic equations using GPU // Computers & Fluids. 2013. Vol. 6. P. 260–268

3. Rostrup S., De Sterck H. Parallel hyperbolic PDE simulation on clusters: Cell versus GPU // Computer Physics Communications. 2010. Vol. 181, No. 12. P. 2164–179

4. Khanna G. High-Precision Numerical Simulations on a CUDA GPU: Kerr Black Hole Tails // Journal of Scientific Computing. 2013. Vol. 56, No. 2. P. 366–380

5. Wong H.C., Wong U.H., Feng X., Tang Z. Efficient magnetohydrodynamic simulations on graphics processing units with CUDA // Computer Physics Communications. 2011. Vol. 182, No. 10. P. 2132–2160

6. LeVeque R. J. Finite volume methods for hyperbolic problems. Cambridge university press, 2002. 588 p.

7. Jose M. Review Article: Seismic modeling // Geophysics. 2002. Vol. 67, No. 4. P. 1304–1325.

8. Gallardo J.M., Ortega S., de la Asuncion M., Mantas J.M. Two-dimensional compact third-order polynomial reconstructions. solving nonconservative hyperbolic systems using gpus // Journal of Scientific Computing. 2011. Vol. 48, No. 1–3. P. 141–163.

9. Castro M. J., Ortega S., de la Asuncion M., Mantas J. M., Gallardo J. M. GPU computing for shallow water flow simulation based on finite volume schemes // Comptes Rendus Mecanique. 2011. Vol. 339, No. 2–3. P. 165–184.

10. Fuhry M., Giuliani A., Krivodonova L. Discontinuous galerkin methods on graphics processing units for nonlinear hyperbolic conservation laws // International Journal for Numerical Methods in Fluids. 2014. Vol. 76, No. 12. P. 982–1003.

11. Mu D., Chen P., Wang L. Accelerating the discontinuous galerkin method for seismic wave propagation simulations using multiple gpus with cuda and mpi // Earthquake Science. 2013. Vol. 26, No. 6. P. 377–393.

12. Komatitsch D. Fluid-solid coupling on a cluster of GPU graphics cards for seismic wave propagation // Comptes Rendus Mecanique. 2011. vol. 339, No. 2–3. P. 125–135.

13. Mielikainen J., Huang B., Huang H., Goldberg M. Improved gpu/cuda based parallel weather and research forecast (wrf) single moment 5-class (wsm5) cloud microphysics // Selected Topics in Applied Earth Observations and Remote Sensing. 2012. Vol. 5. P. 1256–1265.

14. Nickolls J., Buck I., Garland M., Skadron K. Scalable parallel programming with cuda // Queue. 2008. Vol. 6. P. 40–53.

15. Priimak D. Finite difference numerical method for the superlattice boltzmann transport equation and case comparison of cpu(c) and gpu(cuda) implementations // Journal of Computational Physics. 2014. Vol. 278. P. 182–192.

16. Tuttafesta M., Dangola A., Laricchiuta A., Minelli P., Capitelli M., Colonna G. GPU and multi-core based reaction ensemble monte carlo method for non- ideal thermodynamic systems // Computer Physics Communications. 2014. Vol. 185, No. 2. P. 540–549.

17. Ferroni F., Tarleton E., Fitzgerald S. GPU accelerated dislocation dynamics // Journal of Computational Physics. 2014. Vol. 272. P. 619–628.

18. Tamascelli D., Dambrosio F. S., Conte R., Ceotto M. GPU Accelerated Semiclassical Initial Value Representation Molecular Dynamics // ArXiv e-prints. 2013.

19. Karavaev D. A., Glinsky B. M., Kovalevsky V. V. A technology of 3D elastic wave propagation simulation using hybrid supercomputers // Proceedings of the 1st Russian Conference on Supercomputing - Supercomputing Days 2015 Moscow, Russia, September 28-29, 2015. P. 26–33.

20. Khokhlov N. I., Petrov I. B. Application of modern high-performance techniques for solving local and global seismic problems // Proceedings of the 1st Russian Conference on Supercomputing - Supercomputing Days 2015 Moscow, Russia, September 28-29, 2015. P. 380–391.