УДК 519.688

ИНСТРУМЕНТАРИЙ ДЛЯ АВТОМАТИЧЕСКОГО ОПРЕДЕЛЕНИЯ ОПТИМАЛЬНЫХ ЗНАЧЕНИЙ КРИТИЧНЫХ ДЛЯ ПРОИЗВОДИТЕЛЬНОСТИ ПАРАМЕТРОВ ВЫЧИСЛЕНИЙ

Гризан С.А.,
научный руководитель д-р техн. наук Легалов А. И.
Сибирский Федеральный Университет

Один из путей к оптимизации GPGPU-программ заключается в обеспечении максимально возможной загрузки мультипроцессоров GPU на протяжении всего времени работы. Обычно для этого требуется модифицировать GPGPU-ядра таким образом, чтобы соотношение количества вычислительных операций к операциям копирования превышало некоторую величину. Другими словами, необходимо изменить программу так, чтобы вычислительные блоки GPU как можно меньше простаивали, ожидая очередную порцию данных. Для обеспечения данного свойства, например, можно просто увеличивать количество нитей вычислений и правильно разделить их на блоки, что позволит мультипроцессору скрыть обращения к памяти за вычислениями за счёт возможности переключения между блоками, либо же в ручную повышать количество вычислительных операций на одну нить совместно с добавлением асинхронного выполнения операций доступа к глобальной памяти GPU.

К сожалению, хотя данный тип оптимизации и может обеспечить ускорение на десятки процентов, он является крайне не универсальным. Например, при обработке данных другого размера количество или тяжеловесность нитей может существенно измениться, в результате чего мультипроцессоры будут уже не так эффективно загружены. Также при переходе на другую модель GPU с отличным количеством мультипроцессоров и иной пропускной способностью памяти все проведённые оптимизации опять же могут оказаться малополезными.

Решением озвученной проблемы могут стать подходы, основанные на автоподстройке, идея которых заключается во встраивании в программу механизмов, позволяющих непосредственно во время её работы подстраивать GPGPU-ядра под специфику аппаратной платформы, тем самым повышая скорость вычислений. Данная идея была реализована в виде утилиты ttgLib, целью которой является подбор оптимальных значений для любых параметров, которые могут влиять на производительность. Если вернуться к рассмотренному выше подходу к оптимизации путём увеличения количества нитей, то он сводится к двум шагам - (i) модификация GPGPU-ядра и (ii) выбор оптимального разбиения всех нитей на блоки. Во втором шаге как раз и появляются параметры (размер блока для одномерного случая, или же высота и ширина блока для двумерного), поиск оптимальных значений которых можно полностью автоматизировать. Случай же изменения тяжеловесности нити обычно сводится к заданию некоего параметра, определяющего, сколько точек массива должна обработать одна нить, что в большинстве случаев также является параметром.

При использовании утилиты ttgLib все параметры должны быть заменены динамическими параметрами вида Parameter<int>, являющимися шаблонными классами языка C++. В результате данных действий утилита ttgLib получает возможность прозрачно для программы подменять их значения в поиске более оптимальных, для чего используется следующая модель. Время выполнения GPGPU-ядра считается минимизируемой функцией, аргументами которой являются значения тех самых динамических параметров. Таким образом, выполнение GPGPU-ядра с точи зрения утилиты ttgLib является лишь одним замером данной функции времени. При

этом предполагается, что специфика вычислений не сильно изменяется от запуска к запуску одного и того же ядра, что практически всегда верно для нестационарных задач, или же стационарных задач, для решения которых применяются итерационные методы.

Благодаря данной интерпретации, появляется возможность применять различные классические методы оптимизации функционалов, но с рядом ограничений. Во-первых, число доступных замеров достаточно мало (обычно имеется до ста замеров). Во-вторых, функционал времени может изменяться от замера к замеру, в результате чего старые значения могут устаревать. В-третьих, функционал определён не на всей области и может иметь выколотые точки. С учётом данных условий авторами были адаптированы метод покоординатного спуска и генетический алгоритм, первый из которых оказался предпочтительным для простых GPGPU-ядер, использующих до четырёх параметров, а второй — для более сложных ядер.

Апробация разработанной утилиты проводилась на двух модельных задачах аэродинамики, в каждой из которых озвученные параметры подбирались автоматически. В первой задаче использовался только один GPU NVidia Tesla C2050, в то время как во второй расчёты проводились сразу на кластере, каждый узел которого содержал по три GPU NVidia Tesla C2070, что породило дополнительные параметры для балансировки нагрузки.

В первой модельной задаче решалось трёхмерное уравнение Лапласа в постановке задачи Дирихле на структурированной сетке методом Якоби. В данном случае вся область разбивалась на трёхмерные блоки, ширина и высота которых соответствовала ширине и высоте GPGPU-блоков, а глубина которых определяла тяжеловестность нити. Таким образом, путём изменения только трёх данных параметров возможно обеспечить лучшую загрузку всех мультипроцессоров. Также было разработано две версии вычислительных ядер, первое из которых использует разделяемую память, а второе напрямую адресует глобальную память, что может оказаться предпочтительным при наличии кэшей L1/L2. В результате, было получено четыре параметра, три из которых определяли количество нитей и их тяжеловесность, а четвёртая — реализацию ядра.

Тестирование проводилось как на сетках с разным соотношением сторон (каждая из которых поочерёдно увеличивалась в 2/8/16/32/64 раза при сохранении общего количества узлов), так и на сетках разного размера (64x64x64, 96x96x96, 128x128x128). Для минимизации функционала времени выделялось 100 итераций, после чего проводилось ещё 900 итераций с подобранными параметрами. В среднем применение технологии автотюнинга позволило повысить скорость вычислений на 20%, при этом наилучший результат наблюдался на стеке 32x32x2048, на которой было зафиксировано ускорение на 37% — производительность увеличилась с 40 до 55 gigaflops.

Следующей модельной задачей было решение уравнения теплопроводности, заданного на двумерной области с граничными условиями Дирихле с использованием явной разностной схемы. Хотя выбранный алгоритм решения данной задачи практически идентичен методу Якоби, в их программных реализациях есть существенное отличие — из-за двумерной области потенциальный параллелизм в данном случае будет на порядок меньше, в результате чего GPU окажется часто недозагруженным, и, как следствие, его производительность будет достаточно низкой. Чтобы этого избежать, было сделано два вычислительных ядра — первое производит вычисления на GPU, а второе на CPU с использованием расширений SSE, что оказывается более предпочтительным для небольших сеток. Таким образом, было получено три параметра — первый определяет, какое ядро и, соответственно,

вычислитель использовать для расчётов, а второй и третий задают размеры блока при отображении данных на мультипроцессоры GPU.

В аналогии с предыдущей задачей, тестирование проводилось на сетках разного размера. Оказалось, что использование двух CPU Intel Xeon 5650 является более предпочтительным при расчётах на сетках, размер которых не превышает примерно 10 млн. узлов, для больших же сеток вычисления было необходимо переносить на GPU Tesla C2050, так как количество независимых нитей становилось достаточным для его эффективной загрузки.

С помощью рассматриваемой утилиты для автотюнинга ttgLib удалось полностью автоматизировать процесс выбора наиболее подходящего устройства, также как и дополнительно повысить производительность при использовании только GPU. Так, за счёт переключение между CPU и GPU в зависимости от размера сеток в среднем удалось получить ускорение на 50% относительно версии, всегда использующей только один вычислитель. А благодаря подстройке размера блоков скорость GPU реализации алгоритма дополнительно повысилась примерно на 30-40%.

Рассмотренный подход к оптимизации GPGPU-программ путём встраивания в них разработанной авторами технологии автотюнинга оказался достаточно эффективным при решении двух модельных задач из области аэродинамики. При подстройке программы к связке «обрабатываемые данные + аппаратная платформа» за счёт подбора оптимальных значений всего для 4-5 параметров удалось повысить производительность в среднем на 20-60% относительно исходных версий без внесения в них каких-либо существенных изменений.