Зачем что-то считать на GPU?

|  |
| --- |
| Два процессора можно сравнить по разным критериям, наверное, самые популярные — это частота и количество ядер, размер кэшей и прочее, но в конечном счете, нас интересует, сколько операций процессор может выполнить за единицу времени, что это за операции вопрос отдельный, но наиболее распространенной метрикой является количество операций с плавающей запятой в секунду — flops. И когда мы хотим сравнить теплое с мягким, а в нашем случае GPU с CPU, эта метрика приходится как нельзя кстати. |

Архитектура GPU и ее сравнение с CPU

|  |
| --- |
| У видеокарты множество вычислительных ядер, обычно несколько тысяч, но они объединены в блоки, для видеокарт NVIDIA обычно по 32, и имеют общие элементы, в т.ч. и регистры. Архитектура ядра GPU и логических элементов существенно проще, чем на CPU, а именно, нет префетчеров, бранч-предикторов и много чего еще. Еще один важный момент, обычно, видеокарта и процессор не «шарят» память между собой и записать данные на видеокарту и считать результат обратно — это отдельные операции, которые и могут оказаться «бутылочным горлышком» в вашей системе |

Ограничения и возможности при работе с GPU

|  |
| --- |
| Какие ограничения накладывает такая архитектура на выполняемые алгоритмы:   * Если мы выполняем расчет на GPU, то не можем выделить только одно ядро, выделен будет целый блок ядер (32 для NVIDIA). * Все ядра выполняют одни и те же инструкции, но с разными данными, такие вычисления называются Single-Instruction-Multiple-Data или SIMD. * Из-за относительно простого набора логических блоков и общих регистров, GPU очень не любит ветвлений, да и в целом сложной логики в алгоритмах.   Какие возможности открывает:   * Собственно, ускорение тех самых SIMD-вычислений. Простейшим примером может служить поэлементное сложение матриц, его и давайте разберем. |

Приведение классических алгоритмов к SIMD-представлению

|  |
| --- |
| Давайте теперь посмотрим, как будет выглядеть агрегация, приведенная к SIMD представлению: У нас есть массив из n элементов. На первом этапе мы запускаем n/2 потоков и каждый поток складывает по два элемента, т.е. за одну итерацию мы складываем между собой половину элементов в массиве. А дальше в цикле повторяем все тоже самое для вновь получившегося массива, пока не сагрегируем два последних элемента.  Как видите, чем меньше размер массива, тем меньше параллельных потоков мы можем запустить, т.е. на GPU имеет смысл агрегировать массивы достаточно большого размера.  В итоге что мы имеем: множество параллельно работающих потоков, выполняющих один и тот же код, но с разными индексами, а соответственно, и данными, т.е. тот самый SIMD. Это простейший пример, но, если вы хотите работать с GPU, вашу задачу нужно привести к такому же виду. К сожалению, это не всегда возможно и в некоторых случаях может стать темой докторской диссертации, но тем не менее классические алгоритмы все же можно привести к такому виду. |

С чего начать?

|  |
| --- |
| Наиболее распространены две технологии, которые можно использовать для программирования под GPU: OpenCL – это стандарт, который поддерживают большинство производителей видеокарт, в т.ч. и на мобильных устройствах, также код, написанный на OpenCL, можно запускать на CPU. И CUDA – это проприетарная технология и SDK от компании NVIDIA. Сравнивать OpenCL и CUDA несколько не корректно, т.к. одно — стандарт, второе — целое SDK. Тем не менее многие выбирают CUDA для разработки под видеокарты несмотря на то, что технология проприетарная, хоть и бесплатная и работает только на картах NVIDIA. Тому есть несколько причин:   * Более продвинутое API * Проще синтаксис и инициализация карты * Подпрограмма, выполняемая на GPU, является частью исходных текстов основной программы * Собственный профайлер, в т.ч. и визуальный * Большое количество готовых библиотек * Более живое комьюнити |

Результаты выполнения алгоритмов на GPU

|  |
| --- |
| Для теста была взята серверная видеокарта Tesla k80. В примере с агрегацией мы видим существенный прирост производительности с увеличением объема данных. Стоит также обратить внимание на то, что в память карты мы перекачиваем большой объем данных, а назад забираем только одно агрегированное значение, т.е. оверхед на пересылку данных из карты в RAM минимален. |

Оверхед на пересылку данных

|  |
| --- |
| Не всегда очевидно, будет ли GPU эффективен даже в тех задачах, которые хорошо параллелятся. Причиной тому — оверхед на пересылку данных из оперативной памяти компьютера в память видеокарты. Одна из характеристик видеокарты это — memory bandwidth или пропускная способность памяти, которая определяет теоретическую пропускную способность карты. Для Tesla k80 это 480 GB/s. Давайте посмотрим на практические результаты, которые удалось получить для этой видеокарты. Первое, что можно отметить – считывать данные из видеокарты получается быстрее, чем записывать их туда. Второе – при работе с видеокартой можно получить latency от 350 микросекунд, а этого уже может хватить для некоторых low latency приложений. |

Многопоточность

|  |
| --- |
| Теперь давайте посмотрим как будет вести себя видеокарта при обработке нескольких параллельных запросов? Тут приведены сравнения вычислений на GPU и процессоре с 4-мя и 32-мя ядрами. Как видно, процессор справляется с такой нагрузкой очень хорошо, при этом latency для одного запроса на GPU существенно растет с увеличением числа параллельных запросов. |
| На втором графике те же самые вычисления, но с матрицами в 10 раз больше, и GPU под такой нагрузкой ведет себя существенно лучше. Процессор может также довольно эффективно справляться с матричными вычислениями, но до определенных пределов. Для видеокарты характерно то, что для небольшой вычислительной нагрузки производительность падает примерно линейно. С увеличением нагрузки и количества параллельных потоков видеокарта справляется уже лучше. |

Рекомендации

|  |
| --- |
| Если вы размышляете об использовании GPU в своих проектах, то GPU, скорее всего, вам подойдет если:   * Вашу задачу можно привести к SIMD-виду * Есть возможность загрузить большую часть данных на карту до вычислений (закешировать) * Задача подразумевает интенсивные вычисления. |
| Заранее также стоит задаться вопросами:   * Сколько будет параллельных запросов * На какое latency вы рассчитываете * Достаточно ли вам одной карты для вашей нагрузки, нужен сервер с несколькими картами или кластер GPU-серверов. |