Использование numba cuda jit

Москаленко Роман

1 Введение

Игра Новака и Мэя в классическом виде, при синхронной игре и подсчёте очков агентов, позволяет проводить вычисления для агентов отдельно, независимо друг от друга. Значит моделирование игры можно проводить на GPU для большей эффективности.

Пока что были написаны две версии функции использующих numba.cuda.jit и соответственно работающих на GPU. Эти варианты и будут разобраны ниже

2 GPU и cuda

Перед объяснением работы написанных функций, есть несколько основных понятий и принципов, на которых основывается cuda код.

2.1 Блоки и потоки

При каждом запуске cuda функции мы обозначаем количество блоков и потоков в блоках, на которых будут производиться вычисления. Количество блоков и потоков может быть как числом, так и двумерным или трёхмерным массивом, соответственно массивы будут задавать форму сетки блоков и потоков. Каждый поток в блоке будет выполнят весь код написанный в функции, уникальность действий каждого потока достигается за счёт возможности получить номер потока (как внутри блока, так и во всей сетке). Аналогично в случае, когда количество потоков и блоков задаётся массивом, можно узнать координаты потока в заданной сетке. Количество потоков в одном блоке ограничено ≤ 1024

2.2 Память

У GPU есть несколько доступных видов памяти, однако нас интересуют глобальная память(global memory) и разделяемая память (shared

memory).

Глобальная память позволяет нам хранить данные необходимые для вычислений на самом девайсе. Что значительно уменьшает время обращения к переменным, по сравнению с памятью хоста. Желательно перекинуть всё необходимое для вычислений в Глобальную память, это можно сделать вне cuda функции используя cuda.array.

Разделяемая память выделяется отдельно для каждого блока, все потоки в блоке имеют к ней доступ. Её очень мало (от 16 до 40 килобайт.). Время обращения к разделяемой памяти меньше чем к глобальной примерно в 100 раз. Поэтому если поток обращается к чему-то в глобальной памяти несколько раз, или если потоки в блоке обращаются к одним и тем же переменным в глобальной памяти, их стоит подгрузить в разделяемую память и дальше работать с ней.

3 Функция для простой кубической решётки

3.1 Описание алгоритма

Есть главная функция evolve3D_ 5, из которой потом вызываются cuda функции. Тут же просходит выделение глобальной памяти

Две вспомогательные cuda функции нужны для избежания лишних действий с памятью хоста. Первая обнуляет переданный массив, вторая копирует элементы из первого массива во второй. эти функции используются на глобальных массивах. Без них пришлось бы использовать cuda_ to_ device что сильно замедлит программу.

Основной код разбит на две функции culc_ scores3D и new_ strategies3D. Они написаны так, чтобы работать при с трёхмерной сеткой блоков и потоков. Основная идея заключается в том, чтобы каждый блок подгружал, кубический кусочек поля в разделяемую память, затем мы работаем со всеми агентами, кроме крайних, так как для них в разделяемую память загружены все соседи. Блоки смещены так, чтобы соседние блоки пересекались по двум слоям агентов, в итоге крайние агенты каждого блока обрабатываются соседними блоками.

Подгрузка происходит в начале каждой функции. Каждый поток загружает один, соответствующий элемент массива в shared memory. Затем следует команда cuda.sincthreads() которая означает, что потоки прдолжат выполнение программы только, когда все потоки дойдут, до этой метки (то есть когда все потоки подгрузят свой кусочек массива). Далее мы останавливаем все потоки, отвечающие за края блока. Остальной код аналогичен коду в функциях без использования cuda.

В culc_ scores3D мы подгружаем часть массива grid, а в new_ stratagies часть массива scores. Так как для обработки одного агента, каждому потоку в функциях нужно обратиться к соответствующим массивам 27 раз. Остальные массивы подгружать не имеет смысла, так к ним потоки обращаются 1 раз.

3.2 Время работы

Данная функция показала наилучшее время. Она превосходит Cython и Numba.

На кластере благодаря тому, что Общее количество потоков хватает, чтобы запускать большое количество блоков одновременно, время работы почти не зависит от размера поля. При Большом времени эволюции выйгрыш по времени становится особенно большим, так как время не тратится на перенос массивов в глобальную память и обратно.

4 Функция с таблицей соседей

Данная функция работает почти так же долго, как функции с Cython и Numba.

У такой разницы с предыдущей версией функции есть несколько причин.