**Организация параллельных генетических вычислений на GPU**

Основной подход к распараллеливанию генетических алгоритмов для выполнения на GPU состоит в том, чтобы запустить механизм развития последовательным способом на центральном процессоре, и когда новое поколение было бы создано заставить из оцениваться на компьютере с массовым параллелизмом. Предложенный эволюционный алго­ритм достигает ускорения приблизительно в 100 раз. Однако, узким местом может быть в медленной передаче данных от памяти центрального процессора до графиче­ского и назад, особенно для небольших операций.

Алгоритм начинается с инициализации популяции на стороне ЦП. Затем хромосомы и параметры генетического алгоритма передаются в оперативную память *GPU*, используя системную шину. Затем запускается ядро *CUDA*, выполняющее генетический алгоритм для *GPU*. В зависимости от параметров ядра популяции распределяются на несколько блоков (острова) потоков (индивидов). Все потоки на каждом острове читают свои хромосомы из оперативной памяти в быструю память на микросхеме в мультипроцессоре. С этого момента общая память помогает сохранить популяцию острова. Затем процесс развития продолжается с определенным количеством поколений в изоляции, в то время как острова и индивиды эволюционируют на графическом процессоре параллельно. Каждое поколение включает в себя фитнес функцию и отбора индивидов, кроссовера и мутации. Операторы разделены барьерами блока *CUDA* для обеспече­ния целостности данных.



Рисунок 6 – Модель генетического алгоритма на платформе CUDA

Как и в общем случае в параллельном генетическом алгоритмемогут быть использованы различные генетические операторы. Например:

Операторы кросовера:

Кроссинговер

Шаг 1. Выбор родителя 1

Шаг 2. Выбор родителя 2

Шаг 3. Определение места кроссинговера

Шаг 4. Создание потомка 1

Шаг 5. Создание потомка 2

(Рисунокм)

* Арифметический кроссинговер (arithmetical crossover): создаются два потомка:

*H1=(h11,…,hn1), H2=(h12,…,hn2)*,

где *hk1=w\*ck1+(1-w)\*ck2, hk2=w\*ck2+(1-w)\*ck1*, *k=1,…,n, w* либо константа (равномерный арифметический кроссинговер) из интервала [0;1], либо изменяется с увеличением эпох (неравномерный арифметический кроссинговер).

* Геометрический кроссинговер (geometrical crossover): создаются два потомка:

*H1=(h11,…,hn1), H2=(h12,…,hn2),*

где*hk1= (сk1)w\*(ck2)1-w, (ck2)w\*(ck1)1-w, w* – случайное число из интервала [0;1].

* Смешанный кроссинговер (blend, BLX-alpha crossover): генерируется один потомок:

*H=(h1,…,hk,…,hn)*,

где *hk* – случайное число из интервала *[cmin–I\*alpha,cmax+I\*alpha], cmin=min(ck1,ck2), cmax=max(ck1,ck2), I=cmax–cmin.*BLX=0.0 кроссинговер превращается в плоский.

Кроссинговер

{Выбор родителей из популяции}

{Выбор местата для кроссинговера;

Обмен генами (Создание потомков )}

(Рисунокм)

Операторы мутации:

Мутиция

Шаг 1. Выбор особи для мутации

Шаг 2 Определение степени мутации

Шаг 3. Определение места мутации

Шаг 4. Создание мутировавшей особи

Рисунокм

* Одноточечная мутация. Случайно выбирается ген в хромосоме и случайным образом обменивает его на рядом расположенный ген. Схема одноточечной мутации представлена на рисунке 19.

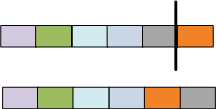


Рисунок 7- Схема одноточечной мутации.

* Двухточечный ОМ заключается в перестановке генов, расположенных справа от точек разрыва.
* Простой оператор мутации. Случайно выбирается ген в хромосоме и случайным образом изменяется. Схема простого оператора мутации представлена на рисунке 20.

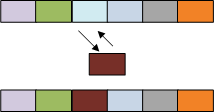


Рисунок 8 - Схема простого оператора мутации.

* Вещественная мутация.

*сi' = ci +* χ*·r·(bi – сi),* если*χ = 0,* или *сi' = ci -* χ*·r·(сi – ai),* если*χ = 1,*

где χ{0, 1} – случайная величина, выбираемая при каждой мутации, которая определяет направление мутации; *r*[0,1] – случайная величина, выбираемая при каждой мутации, которая определяет силу мутации.

Мутация {Выбор особи для популяции}

{Выбор места для мутации;

Создание потомков мутировавшей особи}

Рисунокм

Чтобы смешать генетический материал с разных островов применяется функция миграции. Так как миграция требует межостровной коммуникации используется более медленная оперативная память. Поскольку *CUDA* блоки (острова) не могут быть синхронизированы без потери производительности, процесс миграции выполняется асинхронно, то есть не ждет пока выполнятся предыдущие операции. Это неприемлемо для обычных применений где требуется согласованность данных, но она хорошо рабо­тает на стохастических системах таких как генетический алгоритм.

Выполнение отбора и кроссовера тесно связанны между собой (Рисунок 9).



Рисунок 9 – Схема турнирного отбора с кроссовером

Потоки сгруппированы в пары с помощью общих переменных и барьеров, так чтобы кроссовер мог выполнятся параллельно для целой популяции острова. Во-первых, каждый поток из пары случайным образом выбирает одного родителя, чтобы выполнить кроссовер и фитнес функцию для сравнения. Затем индекс лучшего записывается в общую память (общий массив 1), чтобы уведомит другой поток в паре о более подходящим пар­тнере для выполнения кроссовера.

Процесс миграции представлен на рисунке 10. Острова соединены однонаправленным кольцом, что позволяет острову принимать индивидов от одного соседнего острова. Обмен выполняется асинхронно используя оператив­ную память *GPU*. Число перемещений индивидов определенно параметром *M*.

Во-первых, локальная популяция острова отсортирована согласно фитнес функции. Затем, *M* лучшие индивиды записываются в часть оперативной памяти принадлежа­щей покинутому острову в то время как худшие индивиды *M* записываются как мигранты из оперативной памяти принадлежащей фактическому острову. Сортировка и мигра­ция выполняются параллельно для всех индивидов.



Рисунок 10 – Схема миграции между островами