І.О. Мужиков, магістр, гр. ПІ-45м, ФІКТ
Науковий керівник – д.т.н., проф. кафедри КІ Овезгельдиєв А.О.
Житомирський державний технологічний університет

РЕАЛІЗАЦІЯ ПАРАЛЕЛЬНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

Генетичні алгоритми є потужним предметно незалежним методом пошуку рішення, засновані на теорії Дарвіна. Генетичні алгоритми базуються на таких властивостях живої природи як відбір, мутація і кроссинговер для пошуку точки в просторі станів. Генетичний алгоритм починається з набору особин, що формують популяцію. Зазвичай початкова популяція генерується випадковим чином з використанням рівномірного розподілу. На кожній ітерації алгоритму особину оцінюють за допомогою функції корисності. Алгоритм закінчується, якщо прийнятне рішення було знайдено або обчислювальні ресурси були закінчені. В іншому випадку особин в популяції змінюють шляхом застосування операторів мутації і кросовера. Особин з попередньої популяції називають батьками, особин, які отримуємо шляхом застосування еволюційних операторів називають нащадками.

Хоча генетичні алгоритми дуже ефективні в рішенні багатьох практичних завдань, їх час виконання може стати обмежуючим фактором, так як необхідно обробити дуже багато варіантів рішень. Але більшість трудомістких оцінок придатності може бути виконано не залежно для кожного індивіда в популяції, використовуючи різні методи розпаралелювання.

У наш час виділяють три основних типи паралельних генетичних алгоритмів (ПГА):

Глобальні однопопуляційні ПГА, модель «господар-раб» (Master-Slave GAs); однопопуляційні ПГА (Fine-Grained GAs); багатопопуляційні ПГА (Coarse-Grained GAs).

Модель «господар-раб» характеризується тим, що в алгоритмах такого типу селекція бере до уваги цілу популяцію, на відміну від двох інших моделей. Також варто наголосити на тому, що можливе випадкове схрещування, тобто будь-які два індивіда можуть схрещуватися, в інших моделях схрещування обмежується строго певним набором індивідуумів. В алгоритмах другого класу існує головна популяція, але оцінка цільової функції розподілена серед декількох процесорів. Господар зберігає популяцію, виконує операції ГА і розподіляє індивідууми між підлеглими. Вони ж лише оцінюють цільову функцію індивідуумів. Однопопуляційні ГА придатні для масових паралельних комп'ютерів і складаються з однієї популяції. Селекція і схрещування обмежені відносинами близької спорідненості. Даний клас ПГА може бути ефективно реалізований на паралельних комп’ютерах.

Третій клас – багатопопуляційні ГА більш складна модель, так як вона складається з декількох підпопуляцій, які періодично, за встановленими правилами, обмінюються індивідуумами. Такий обмін індивідуумами називається міграцією і управляється кількома параметрами. Багатопопуляційні ГА дуже популярні, але достатньо складні як для розуміння так і для реалізації, тому що наслідки від ефекту міграції, на даний момент, в повному обсязі не досліджені. У той же час багатопопуляційні ГА мають схожість з «острівною моделлю» в популяційної генетики, яка розглядає відносно ізольовані громади.

Вона є однією з найбільш перспективних варіантів багатопопуляційного ПГА. Так, як острівна модель може в повній мірі використовувати всю доступну обчислювальну потужність комп'ютерів. При використанні даної моделі популяції діляться на острови, їх ще можна назвати підпопуляції, і кожна з них розвивається окремо один від одного. Острови популяції вільно сходяться до різних оптимумів. Застосування оператора міграції дозволяє змішувати хороші риси, які виникають на різних островах популяцій.

В даний час сучасні графічні прискорювачі, розроблені для прискорення 3d рендеринга в режимі реального часу, можна розглядати як систему з високим ступенем паралелізму, і, отже, можна з користю застосовувати їх для прискорення розрахунку генетичних алгоритмів.

Графічні процесори еволюціонували в дуже потужні й гнучкі процесори, це обумовлено постійно зростаючим вимогам індустрії відеоігор. Зараз вони проводять обчислення з плаваючою комою набагато швидше, ніж сьогоднішні центральні процесори. Також, крім графічних додатків, вони дуже добре підходять для вирішення спільних проблем, які можуть бути виражені у вигляді паралельного обчислення (тобто один і той же код може виконуватись з різними елементами даних).

Більш того, кілька загально цільових мов високого рівня для графічних процесорів стали доступні в таких технологіях як CUDA і OpenCL. Таким чином, розробникам не потрібно більше освоювати додаткових методів програмування графічних процесорів через API-інтерфейси.

Сучасні відеокарти є насправді дуже потужними масивно-паралельними комп'ютерами з єдиним головним недоліком: всі елементарні процесори на карті організовуються в більші мультипроцесори. Вони повинні виконати одну й ту ж саму інструкцію одночасно, але з різними потоками даних.

Генетичні алгоритми необхідні для виконання однакових функції оцінки з різними індивідами (що можна розглядати як різні дані), це і наводить на думку про те, що застосування графічних процесорів в розрахунках допоможе в їх прискоренні.

Основна ідея, яка спадає на думку, коли необхідно розпаралелити еволюційний алгоритм полягає в тому, щоб запустити механізм розвитку послідовним способом на центральному процесорі, і коли нове покоління було б створено, змусити його оцінюватися на комп'ютері з масовим паралелізмом. Запропонований еволюційний алгоритм досягає прискорення приблизно в 100 разів. Однак, слабким місцем може бути повільна передача даних від пам'яті центрального процесора до графічного і назад, особливо для невеликих операцій.

Для реалізації генетичного алгоритму на графічному процесорі (GPU) була обрана платформа CUDA. Цей інструмент обіцяє досягти високого прискорення на GPU. CUDA додаток може виконуватися на будь-якій відеокарти NVIDIA починаючи з GeForce 8 як на Linux системах так і на Windows.

Як зазначалося раніше NVIDIA GPU складається з мультипроцесора, здатного виконувати завдання паралельно. Потоки, що працюють в цих процесорах дуже маленькі і здатні синхронізуватися, використовуючи бар'єри, так щоб цілісність даних зберігалася. Це може бути зроблено з дуже низьким впливом на продуктивність в мультипроцесорі, але не між мультипроцесорами. Це обмеження змушує нас створювати острови або абсолютно незалежні, або виконувати міграцію між ними асинхронно.

Пам'ять, яка використовується в відеокартах, розділена на два рівня - оперативна пам'ять і пам'ять на мікросхемі. Оперативна пам'ять має велику ємність (сотні Мбайт) і містить повний набір призначених для користувача програм, а також даних. На жаль, велика ємність переважується з високою затримкою. З іншого боку, пам'ять на мікросхемі дуже швидка, але має дуже обмежений розмір. Крім локальних регістрів на кожному потоці, пам'ять мікросхеми містить особливо корисні спільно використовувані сегменти мультипроцесора. Розмір пам'яті на мікросхемі – строго обмежує фактор для розробки ефективного генетичного алгоритму, але існуючі програми CUDA значною мірою отримують від цього користь.

Ідея полягає у створенні островів, з міграцією уздовж однонаправленного кільця. Кожним індивідом керує єдиний потік CUDA. Популяції зберігаються в пам'яті спільного користування на мікросхемі на певних мультипроцесорах GPU (блоки CUDA). Це гарантує і інтенсивне виконання в обчислювальному відношенні, і великий паралелізм, необхідний для GPU.

Запропонований алгоритм починається з ініціалізації популяції на стороні ЦП. Потім хромосоми і параметри генетичного алгоритму передаються в оперативну пам'ять GPU, використовуючи системну шину. Потім запускається ядро CUDA, яке виконує генетичний алгоритм для GPU. Залежно від параметрів ядра популяції розподіляються на кілька блоків (острови) потоків (індивідів). Всі потоки на кожному острові читають свої хромосоми з оперативної пам'яті в швидку пам'ять на мікросхемі в мультипроцесорі. З цього моменту спільна пам'ять допомагає зберегти популяцію острова. Потім процес розвитку триває з певною кількістю поколінь в ізоляції, в той час як острови і індивіди еволюціонують на графічному процесорі паралельно. Кожне покоління включає в себе фітнес функцію і відбір індивідів, кросовера і мутації. Оператори розділені бар'єрами блоку CUDA для забезпечення цілісності даних.

Щоб змішати генетичний матеріал з різних островів застосовується функція міграції. Так як міграція вимагає міжострівної комунікації використовується більш повільна оперативна пам'ять. Оскільки CUDA блоки (острови) не можуть бути синхронізовані без втрати продуктивності, процес міграції виконується асинхронно, тобто не чекає поки будуть виконуватися попередні операції. Це неприйнятно для звичайних застосувань де потрібна узгодженість даних, але вона добре працює на стохастичних системах таких як генетичний алгоритм. Виконання турнірного відбору і арифметичного кросовера тісно пов'язані. Потоки (індивіди) згруповані в пари за допомогою загальних змінних і бар'єрів, так щоб кросовер міг виконуватися паралельно для цілої популяції острова. По-перше, кожен потік з пари випадковим чином вибирає одного з батьків, щоб виконати кросовер і фітнес функцію для порівняння. Потім індекс кращого записується в загальну пам'ять (загальний масив 1), щоб повідомить інший потік в парі про більш відповідним партнері для виконання кросовера. Далі виконується паралельна генерація випадкових чисел на всьому острові і результати записуються в загальний масив 2. Потім пара потоків шукають своє випадкове число з цього масиву і порівнює його з ймовірністю кросовера, щоб вирішить виконувати кросовер чи ні. Це завдання споживає першу половину масиву 2. Друга половина використовується під час арифметичного кросовера як ваги агрегації:

$$O_1 = a \cdot P_1 + \ 1 - a \ \cdot P_2 \quad (1)$$

$$O_2 = \ 1 - a \ \cdot P_1 + a \cdot P_2 \quad (2)$$

де O_1 і O_2 представляють нащадків, P_1 і P_2 представляють батьків і позначають ваги агрегації. Такий підхід марнує вибір індивідів в разі невиконання кросовера.

У процесі міграції острови з'єднані односпрямованим кільцем, що дозволяє острову приймати індивідів від одного сусіднього острова. Обмін виконується асинхронно використовуючи оперативну пам'ять GPU. Число переміщень індивідів безумовно параметром M. По-перше, локальна популяція острова відсортована відповідно до фітнес функції. Потім, M кращих індивідів записуються в частину оперативної пам'яті, яка належить покинутому острову, в той час як найгірші індивіди M записуються як мігранти з оперативної пам'яті, які належать фактичному острову. Сортування і міграція виконуються паралельно для всіх індивідів. Для реалізації подібного алгоритму необхідно застосувати як мінімум бібліотеку NVIDIA CUDA Toolkit 7.5 і Visual Studio 2012. Для проведення повномасштабних експериментів необхідно використовувати графічні прискорювачі від компанії NVIDIA, з різною кількістю ядер CUDA.