The article deals with modern architectures used in video adapters. The code is given as an example, which is used in working with graphical process units, and the instructions how to work with the global memory are offered. It could be concluded that such a technology can be used in a wide variety of applications and can be developed in different ways. Different patterns of thread interaction in this technology are shown. The problems of synchronization and different solutions are considered. The dynamic parallelism and its principles are defined.

A serial version of the Gaussian elimination (solving systems of linear algebraic equations) and its parallel version on the CUDA architecture cores in the Python programming language were implemented. A number of research procedures were done: a comparison of the speed of the various implementations of the Gaussian elimination with using of various program libraries and platforms: Anaconda (Numba), PyCUDA, KappaCUDA, and PyOpenCL.

Keywords: GPGPU, CUDA, SIMD, SIMT, Python, PyCUDA, Anaconda, PyOpenCL, KappaCUDA, Gaussian elimination.

Матеріал надійшов 13.10.2017

УДК 004.421.2:519.17

Глибовець М. М., Петльована М. В., Кирієнко О. В.

ЗАСТОСУВАННЯ ЕВОЛЮЦІЙНИХ АЛГОРИТМІВ ДЛЯ РОЗВ'ЯЗАННЯ ЗАДАЧІ АПРОКСИМАЦІЇ ЗОБРАЖЕНЬ МНОГОКУТНИКАМИ

У статті описано розробку та реалізацію еволюційного алгоритму розв'язку задачі апроксимації зображення многокутниками, запропоновано структуру даних для ефективного кодування зображень (зі змінною кількістю рядків та стовпців зображення, кількістю многокутників в одній клітинці, кількістю точок многокутників та ступенем їх перетинання). Було впроваджено стратегії мутації для виведення розв'язку з локального оптимуму та подальшого знаходження глобального оптимуму. Для пришвидшення роботи алгоритму етапи селекції та перевірки критерію завершення було реалізовано з використанням моделі розподілених обчислень MapReduce.

Ключові слова: еволюційні алгоритми, задачі апроксимації зображення многокутниками, MapReduce.

Вступ

Представлення зображень у вигляді множини многокутників є важливим етапом для багатьох задач аналізу зображень, як-от розпізнавання об'єкта, зіставлення зображень, супровід цілі і т. п. Актуальність та важливість цих задач у сучасній науці та техніці створює необхідність розробки ефективних алгоритмів апроксимації зображень многокутниками. Еволюційні алгоритми є перспективним напрямом вирішення поставленої проблеми.

У задачі генерації зображень за допомогою еволюційних алгоритмів можна виділити два основні напрями: автоматизована генерація зображень та оптимізація і покращення якості готових зображень. Генетичні технології ефективно допомагають у реконструкції і стилізації зображень, а саме в покращенні освітлення, контрастності, чіткості контурів тощо.

Проблемою апроксимації зображень та графічних об'єктів за допомогою многокутників займалося багато вчених світу. Зокрема, у роботах [4; 5] розглянуто ефективні алгоритми для наближення

полігонами кривих та розкладу двовимірних графічних об'єктів у набір опуклих многокутників.

Апроксимація зображень критично важлива для можливості значного зменшення кількості точок та графічних примітивів, необхідних для опису зображення без значних втрат для деталізації і точності зображення з метою покращення характеристик зображення для їх подальшої обробки. Це значно підвищує ефективність методів розпізнавання образів, сегментації зображення, видалення шуму.

Метою цієї роботи є опис створеного еволюційного алгоритму та його програмної реалізації для розв'язання задачі пошуку наближеного представлення зображень множиною многокутників. Для досягнення цієї мети потрібно було насамперед: розробити структуру даних для ефективного кодування зображення; підібрати та налаштувати параметри еволюційного алгоритму, за яких формується прийнятне рішення відповідно до критеріїв оптимізаційної задачі, що формалізує задачу наближення зображення многокутниками; визначити генетичні оператори, які застосовуватимуться для створення/модифікації особин популяції. Основу розв'язку склало еволюційне моделювання динаміки змін і аналіз впливу того чи іншого фактора на змодельовану проблему.

Як інструмент створення програмного засобу було обрано PyCharm IDE 2017.1.2 – інтегроване середовище розробки (IDE) мовою програмування Python, яке є безкоштовним, вільнопоширюваним, із відкритим вихідним кодом, та спеціалізовані бібліотеки: NumPy – надає підтримку великих багатовимірних масивів і матриць, разом із великою бібліотекою високорівневих математичних функцій для операцій з цими масивами; Python Imaging Library (PIL) – для роботи з растровою графікою; Joblib – надає набір інструментів для паралельного програмування.

Постановка задачі

Під словом «зображення» надалі будемо розуміти матрицю:

$$A = \{a_{ij}\}_{i=\overline{1,m},\, j=\overline{1,n}},$$

де m та n – висота та ширина зображення відповідно;

a_{ij} – це ціле число з множини $\{0,1\}$.

Такому типу матриці відповідають монохромні (або двійкові) чорно-білі зображення.

На вхід алгоритму задамо деяке еталонне зображення E, яке потрібно наблизити многокутниками. Задача полягає у відшуканні такого зображення F, яке буде сформоване за допомогою деякої множини многокутників, кожен із яких матиме свій колір, і яке буде максимально подібним до еталонного.

Для того щоб оцінити ступінь схожості зображень, потрібно ввести функцію порівняння зображень – для цього можна застосувати середньоквадратичне відхилення згенерованого зображення від еталонного.

Згенероване зображення F можна також представити в іншому вигляді:

$$F = \{M_i\}_{i=\overline{1,k}},$$

де M_i – многокутник, що задається множиною точок та кольором.

Тобто потрібно створити еволюційний алгоритм, що може згенерувати таку множину многокутників, відмалювання яких сформувало б у результаті таке зображення, яке було б максимально подібним до еталонного.

Отже, маємо справу з класичною задачею оптимізації, цільовою функцією якої виступає мінімізація середньоквадратичної різниці згенерованого та еталонного зображень, а обмеженнями виступатимуть параметри еволюційного алгоритму, які буде розглянуто нижче.

Нагадаємо, що генетичні алгоритми – це процедури пошуку, засновані на механізмах природного відбору і спадкоємства [2]. У них використовується еволюційний принцип виживання найбільш пристосованих особин. Вони відрізняються від традиційних методів оптимізації декількома базовими елементами. Зокрема, генетичні алгоритми обробляють не значення параметрів самого завдання, а їх закодовану форму; здійснюють пошук рішення, виходячи не з єдиної точки, а з їх деякої популяції; використовують тільки цільову функцію, а не її похідні або іншу додаткову інформацію; застосовують імовірнісні, а не детерміновані правила вибору.

Перераховані чотири властивості, які можна сформулювати також як кодування параметрів, операції на популяціях, використання мінімуму інформації про завдання і рандомізація операцій, приводять у результаті до стійкості генетичних алгоритмів і до їхньої переваги над іншими широковживаними технологіями.

Важливою особливістю генетичних алгоритмів є те, що вони працюють не з одним, а з множиною альтернативних розв'язків, які в процесі роботи алгоритму певним чином змагаються між собою за «виживання». Після завершення роботи алгоритму обирається найкращий розв'язок, який і вважається результатом роботи алгоритму.

Важливим моментом програмної реалізації має бути можливість налаштовування структури даних для генерації полігонів, що надасть широкі можливості для дослідження апроксимації зображення за допомогою різних многокутників – трикутників, п'ятикутників тощо.

Алгоритм апроксимації зображень многокутниками та його реалізація

Для розв'язання поставленої задачі було розроблено структуру даних, яка може ефективно представляти зображення у вигляді послідовності многокутників. Так звана канва зображення розграфлюється на клітини однакового розміру. У кожній клітині канви міститься певна, задана налаштуваннями, кількість многокутників, кожен з яких задається списком точок та кольором. Програмну реалізацію структури даних наведено на рис. 1.

```
classCellPolygon(object):
def__init__(self, i_manifest):
self.manifest = i_manifest
val = randint(0, 1)
self.color = [val, val, val, 1] #rgba
self.points = []
for i inrange(i_manifest.POLYGONS_POINTS_
NUMBER):
self.points.append([uniform(0, 1), uniform(0, 1)])

defmutate(self, i_mutation_ratio=0.05):
rand_number = random()
ifrand_number > 0.5:
        #mutatecolor
ifself.color[0] == 1:
self.color[0] = 0
self.color[1] = 0
self.color[2] = 0
else:
self.color[0] = 1
self.color[1] = 1
self.color[2] = 1
else:
        #mutatepoints
for p inself.points:
new_point_coord = uniform(0, 1)
whilemath.fabs(p[0] – new_point_coord) \
<i_mutation_ratio:
new_point_coord = uniform(0, 1)
            p[0] = new_point_coord
new_point_coord = uniform(0, 1)
whilemath.fabs(p[1] – new_point_coord) \
<i_mutation_ratio:
new_point_coord = uniform(0, 1)

            p[1] = new_point_coord

classPolygonedImage(object):
def__init__(self):
self.cells = []

defgenerate(i_manifest):
image = PolygonedImage()
for i inrange(i_manifest.ROWS_NUMBER):
image.cells.append([])
for j inrange(i_manifest.COLS_NUMBER):
image.cells[i].append([])
for k inrange(i_manifest.POLYGONS_PER_CELL):
image.cells[i][j].append(CellPolygon(i_manifest))

returnimage

classChromosomesPool(object):
defgenerateChromosomes(i_number, i_manifest):
polygoned_images = []
for i inrange(i_number):
polygoned_images.\
append(PolygonedImage.generate(i_manifest))
returnpolygoned_images
```

Рис. 1. Структура даних представлення зображення

Для налаштування параметрів перед початком виконання алгоритму було введено маніфест структури даних, у якому можна вказати налаштування, що цікавлять дослідника (зразок маніфесту наведено на рис. 2): кількість рядків m' і стовпців n', на які буде розграфлено зображення; кількість многокутників k в одній клітинці зображення; кількість точок p кожного многокутника; максимально можливий ступінь перекривання многокутників.

```
classManifest(object):
def__init__(self, i_img_width, i_img_height):
self.IMG_WIDTH = i_img_width
self.IMG_HEIGHT = i_img_height
self.ROWS_NUMBER = 4
self.COLS_NUMBER = 4
self.POLYGONS_PER_CELL = 8
self.POLYGONS_POINTS_NUMBER = 5
self.POLYGONS_OVERLAPPING = 1.2
self.ROW_HEIGHT = i_img_height / self.ROWS_
NUMBER
self.ROW_WIDTH = i_img_width / self.COLS_
NUMBER
```

Рис. 2. Приклад маніфесту структури даних

Особиною популяції є зображення, представлене описаною вище структурою даних. Геном хромосоми виступає многокутник. Додаткове кодування не застосовується, таким чином, хромосомою є список многокутників, кожен з яких є списком точок та змінною, що позначає колір. Координати точок є парами дійсних чисел (x, y), причому для зручності $x, y \in [0, 1]$ – під час малювання многокутників відрізок $[0, 1]$ масштабується на відрізок $[-1, 1]$ для того, щоб прив'язати малювання многокутника до центру відповідної клітини. Отже, під час малювання координата точки (x, y) перетвориться у відповідну координату на канві зображення:

$$(i + 0.5 + (2 * x - 1) / 2 * k_{overlap}) * w, (j + 0.5 + (2 * y - 1) / 2 * k_{overlap}) * h),$$

де i, j – номер рядка і стовпчика відповідної клітини сітки;

w, h – ширина і висота клітини сітки;

$k_{overlap}$ – коефіцієнт накладання многокутників.

Як було описано у [2], налаштування параметрів алгоритму, зокрема розміру популяції, суттєво впливає на роботу алгоритму. Дійсно, емпірично було встановлено, що за меншого розміру популяції ($N = 50$) алгоритм передчасно збігається до локального оптимуму, внаслідок того, що особини популяції покривають занадто малу частину пошукового простору. Застосування оператора мутації виявляється недостатнім для виведення популяції з поточного розв'язку, при цьому не має значення, наскільки мутуватиме особина (перевірено для граничних параметрів $p_m = 0.0025$ та $p_m = 0.05$), оскільки новоутворена особина показує гірший коефіцієнт здоров'я (незважаючи на те, що такий розв'язок може містити частину оптимального розв'язку) і загине на етапі відбору в наступне покоління. Водночас збільшення розміру популяції призводить до квадратичного збільшення часу роботи алгоритму і використаної пам'яті для відмалювання і порівняння особин популяції. Однак це дає змогу досягти більшої різноманітності генотипу в популяції, тому розмір популяції був встановлений рівним $N = 200$, а витрати обчислювальних ресурсів компенсуються застосуванням паралельних і розподілених обчислень.

Для відбору особин для генерації нащадків використовується панміксія, а для заміщення – елітарний відбір. Функцією пристосованості є цільова функція, сама ж пристосованість особини обчислюється як середньоквадратичне відхилення згенерованого зображення від еталонного. Для схрещування застосовується дискретна рекомбінація. Оператор мутації для кожного гена, що мутує, з рівною імовірністю обирає, що саме підлягатиме мутації – колір многокутника чи точки. Для бінарних зображень мутація кольору многокутника є, по суті, інверсією, а мутація координат точки здійснюється в її m-околі точки, де m – коефіцієнт мутації. Для підтримання різноманітності популяції застосовується метод вилучення дублікатів.

Налаштування всіх перелічених параметрів дає змогу генерувати зображення, проте через якийсь час популяція потрапляє в локальний оптимум і досягає стану адаптації. Подальші ітерації алгоритму не можуть вивести популяцію з локального оптимуму, саме тому подальшим кроком удосконалення алгоритму стало введення адаптивного керування параметрів, а саме: впровадження стратегій мутації (стратегії, застосовані для розв'язання задачі, наведено на рис. 3).

```
classMutationConfiguration(object):
def__init__(self,
i_mutate_parents,
i_minimum_mutation_rate,
i_mutation_count,
i_leave_parents=True):
self.mutate_parents = i_mutate_parents
self.minimum_mutation_rate = i_minimum_mutation_rate
self.leave_parents = i_leave_parents
self.mutation_count = i_mutation_count

classMutationStrategy(object):
def__init__(self):
self.configurations = [
MutationConfiguration(True, 0.0025, 0.1),          #0
MutationConfiguration(False, 0.0025, 0.1, False),  #1

MutationConfiguration(True, 0.0025, 0.25),         #2
MutationConfiguration(False, 0.0025, 0.25, False), #3

MutationConfiguration(True, 0.005, 0.1),           #4
MutationConfiguration(False, 0.005, 0.1, False),   #5

MutationConfiguration(True, 0.005, 0.25),          #6
MutationConfiguration(False, 0.005, 0.25, False),  #7
    ]
```

Рис. 3. Стратегії мутації

Якщо немає покращення розв'язків протягом n ітерацій (на практиці n було покладено рівним 10), тоді відбувається зміна стратегії мутації. Кожна стратегія мутації задається власною конфігурацією, що складається з чотирьох параметрів:

– булевий параметр, що вказує, чи потрібно перед етапом заміщення мутувати батьків нової популяції. Після застосування генетичних операторів новоутворені особини можуть бути більш пристосованими для розв'язання задачі, тому на етапі заміщення їхні батьки просто будуть вилучені з популяції. Однак, спираючись на знання про контекст задачі, можна припустити, що насправді батьки нових особин є ближчими до розв'язання задачі, але для цього потребують певних змін (наприклад, якщо застосувати інверсію та мутацію до декількох полігонів, згенероване зображення буде більш подібним до оригінального);

– коефіцієнт мутації – на початку виконання алгоритму точки полігонів зазнають незначної мутації, проте, коли популяція адаптується, потрібно збільшити вплив оператора мутації, щоб вивести популяцію з локального оптимуму;

– кількість особин популяції, що підлягатимуть мутації – подібно до коефіцієнта мутації, цей параметр зростає разом зі зростанням адаптації популяції;

– булевий параметр, що вказує, чи потрібно вилучити батьків після породження нащадків – навіть якщо батьки матимуть кращі показники пристосованості і відповідно до стратегії відбору особин заслуговуватимуть на продовження життя в популяції, якщо стратегією вказано, що батьки мають бути вилученими з популяції, – вони видаляються. Таким чином, алгоритм дотримується принципу «крок назад і два вперед»: після вилучення батьків із пулу особин дуже ймовірно, що загальний рівень здоров'я популяції погіршиться, проте це дасть змогу гіршим особинам затриматися в популяції і дати потомство, яке, своєю чергою, матиме краще значення пристосованості, ніж пращури.

Методи вирішення побічних проблем при апроксимації зображень за допомогою еволюційних алгоритмів

При апроксимації зображень многокутниками найбільш ресурсоємним завданням є відмалювання зображення, представленого множиною многокутників, для порівняння з еталонним зображенням, та порівняння особин популяції для відбору хромосом у наступне покоління. Зрозуміло, що чим більша розмірність зображення, тим більше часу потрібно на виконання цих дій. Саме тому для того, щоб зменшити час виконання цих етапів програми, пропонується застосувати парадигми паралельного програмування та використати модель розподілених обчислень MapReduce.

Для розпаралелювання відмалювання зображень на ядра обчислювальної машини використовується модуль багатопроцесорної обробки для обчислення паралельного застосування функції до багатьох різних аргументів. Після того, як множина многокутників була перетворена на зображення, здійснюється обрахунок функції пристосованості зображень таким чином: будується зображення, що є різницею між зображенням-особиною популяції та еталонним зображенням, після чого будується статистична гістограма різниці, що повертається у вигляді списку значень пікселів зображення-різниці.

Під час відмалювання зображень важливим моментом є позиція многокутника у відповідній клітині сітки. Якщо генерацію точок многокутника обмежувати лише границями відповідної клітини, це призведе до того, що проміжки між клітинами будуть не заповненими, а отже, зображення, отримане з такої сітки, не зможе наблизитися до оригінального зображення (наприклад, на рис. 4 неозброєним оком видно границі сітки).

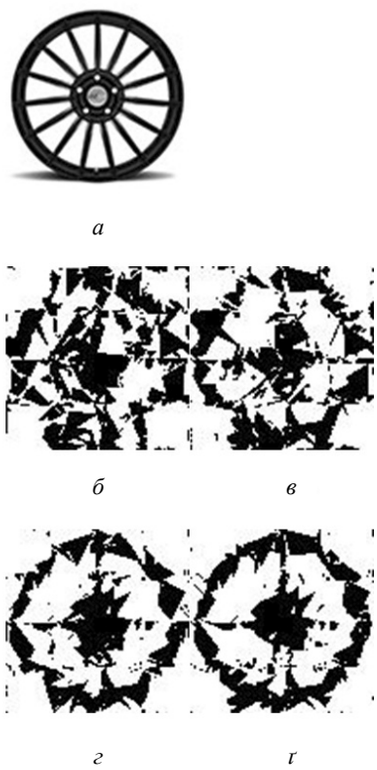

Рис. 4. Апроксимація зображення колеса з сіткою 2 х 2 і без накладання многокутників за межами клітин: *а* – еталонне зображення колеса; *б* – 30-те покоління; *в* – 50-те покоління; *г* – 100-те покоління; *ґ* – 400-те покоління популяції

Саме тому було введено коефіцієнт накладання многокутників, що, по суті, розширює межі клітини, де буде згенеровано многокутники, на деякий коефіцієнт. Таким чином, накладання многокутників закриє щілини між клітинами сітки та візуально не виявлятиме внутрішню структуру-сітку зображення.

Висновки

У роботі описано розробку і реалізацію еволюційного алгоритму для задачі апроксимації зображення многокутниками. Для ефективного кодування зображень було запропоновано структуру даних зі змінною кількістю рядків та стовпців зображення, кількістю многокутників в одній клітинці, кількістю точок многокутників та ступенем їх перетинання. Було впроваджено стратегії мутації для виведення розв'язку з локального оптимуму та подальшого знаходження глобального оптимуму. Для пришвидшення роботи

алгоритму етапи селекції та перевірки критерію завершення реалізовано з використанням моделі розподілених обчислень MapReduce.

Проведені дослідження підтверджують перспективність використання еволюційних алгоритмів для розв'язання задач обробки зображень та генерації нових зображень. Зокрема, можна запропонувати такі поліпшення алгоритму:

1) виділити маску важливості областей зображення – деяке зображення з виділеними областями зацікавленості, яке можна використати при розрахунку функції пристосованості, що дасть змогу зменшити шум у популяції і сконцентрувати пошуки на галузі зацікавленості дослідника [1];

2) продовжувати налаштовувати параметри алгоритму – підібрати оптимальні параметри алгоритму (зокрема, налаштування мутацій у стратегіях мутацій) та застосувати самоадаптацію параметрів під час еволюції;

3) застосувати модель паралельного еволюційного алгоритму, наприклад, острівну модель, що дозволяє запустити алгоритм відразу кілька разів і поєднати «досягнення» різних прогонів-островів для отримання найкращого рішення [3];

4) змінивши функцію порівняння згенерованого зображення з еталонним на інтерактивну взаємодію з користувачем, представлений алгоритм можна перетворити на алгоритм генерації фоторобота.

Список літератури

1. Аппроксимация изображений генетическим алгоритмом при помощи EvoJ [Електронний ресурс] // Хабрахабр. – Режим доступу: https://habrahabr.ru/post/149161/. – Назва з екрана.
2. Глибовець М. М. Еволюційні алгоритми : підручник / М. М. Глибовець, Н. М. Гулаєва. – Київ : НаУКМА, 2013. – 828 с.
3. Острівна модель [Електронний ресурс] // Вікіпедія – вільна енциклопедія. – Режим доступу: https://uk.wikipedia.org/wiki/Острівна_модель. – Назва з екрана.
4. Huang S.-C. Polygonal approximation using genetic algorithms / Shu-Chien Huang, Yung-Nien Sun // Pattern Recognition. – 1999. – Vol. 32, no. 8. – P. 1409–1420.
5. Jadhav S. H. Polygonal Approximation of 2-D Binary Images / Seema H. Jadhav, B. S. Tarle, L. M. Waghmare // Third International Conference on Information Technology: New Generations. – Las Vegas, Nevada, USA, 2007. – P. 219–223.

M. Glybovets, M. Petliovana, O. Kyriienko

USING EVOLUTIONAL ALGORITHMS FOR SOLVING THE PROBLEM OF APPROXIMATION OF IMAGES BY POLYGONS

This article describes a development and implementation of an evolutionary algorithm for solving the problem of image approximation by polygons, a proposed data structure for efficient image coding (with the variable number of image rows and columns, polygons in one cell, polygons points and their intersection degree), an implemented mutation strategies for solving output from a local optimum and further global optimum finding. The selective and verifiable stages of a completion criterion were implemented using the MapReduce by distributing computing model to speed up the algorithm running.

The conducted research confirms a perspective of the evolutionary algorithms usage for solving image processing and generating new images problems. In particular, we can propose such algorithm improvements:

1) to highlight a mask of importance of the image areas – a certain image with the accented areas of interest that can be used for the fitness function calculation, which will reduce the noise in the population and concentrate searches on the explorer's area of the interest;

2) to continue setting up the algorithm parameters – to pick up the optimal algorithm parameters (particularly, setting up mutations in mutation strategies) and to apply the parameter self-adaptation during the evolution;

3) to apply a parallel evolutionary algorithm model, for instance, an island model that allows one to run the algorithm several times at once and to combine the "achievements" of different runs – islands for getting the best solution;

4) the presented algorithm can be transformed into photo-robot generation algorithm by changing the function of comparing the generated image with a benchmark to the interactive user.

Keywords: evolutional algorithm**,** the problem of image approximation by polygons, MapReduce.

Матеріал надійшов 12.10.2017