ЦИФРОВА ЕКОНОМІКА ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ
Digital Economy and Information Technologies

УДК 004.272

Дмитро Ушенін,
Сергій Сабанов,
Ольга Шляга

Запорізький інститут економіки та інформаційних технологій
Запоріжжя, Україна

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ВИКОРИСТАННЯ ТЕХНОЛОГІЙ GPGU В КАСТОМНИХ ПЛАГІНАХ ADOBE AFTER EFFECTS

Анотація. У роботі розглядаються особливості використання технологій паралельних обчислень на базі графічної підсистеми комп’ютера в плагінах Adobe After Effects від сторонніх розробників. Аналіз показав, що під час створення плагінів для середовища Adobe After Effects розробники досить часто ускладнюють код, намагаючись перекласти розпаралелені обчислення з традиційних засобів центрального процесора на функціональні блоки GPU, що на сучасному етапі вважається безумовно правильним. Однак, при детальному розгляді виявляється, що ефективність, яку користувач отримує від роботи таких плагінів, аж ніяк не вища за ефективність роботи плагінів, що мають значно простіший код, що традиційно працює на CPU. Така ситуація обумовлена відсутністю аналізу особливостей роботи розпаралеленого коду в середовищі графічної підсистеми під час обробки різних візуальних ефектів, і, як слідство, відсутності чітких рекомендацій щодо необхідності, чи, навпаки, недоцільності роботи з апаратною частиною GPU в коді. В даній статті наведено дослідження факторів, які впливають на ефективність використання паралельних обчислень на GPU у плагінах Adobe After Effects. Даються рекомендації доцільності використання GPU в залежності від спеціалізації плагіну, пов’язаної з обробкою того чи іншого візуального ефекту

Ключові слова -*CPU, GPU, CUDA, OPENCL, Adobe After Effects, паралельні обчислення, плагін*

1. ВСТУП

Adobe After Effects – одна з найпопулярніших сучасних програм для редагування відео, створення анімацій та різноманітних ефектів. Застосунок має більш ніж достатній функціонал, який, тим не менш, може бути значно розширений за допомогою сторонніх інтеграцій, серед яких: плагіни, скрипти та розширення. Такий механізм реалізації додаткових можливостей можна вважати основним для більшості сучасних програм.

Плагіни для середовища переважно написані на C або C++ і розширюють функціональність Adobe After Effects, дозволяючи використовувати більш складні функції, зокрема такі, як системи частинок, механізми фізики, 3D-ефекти тощо. Приклади найбільш розповсюджених готових плагінів: Element 3D [1], Optical Flares [2], Magic Bullet Looks [3], Boris Continium Complete [4, 5] та ін.

У разі вирішення якихось нетипових завдань доцільним є самостійне написання плагінів, що забезпечують потрібний функціонал. Під час створення таких спеціалізованих модулів є можливість використання паралельних обчислень на GPU, що при певних обставинах може зменшити час обробки відео у декілька разів. Adobe After Effects підтримує

наступні технології використання GPGPU: CUDA, OpenCL, OpenGL та Metal [6], кожна з яких може бути доступна в залежності від операційної системи, встановленої відеокарти та деяких інших факторів.

Детальний розгляд особливостей застосування GPGPU показує, що використання графічного ядра в плагінах далеко не завжди є ефективним і підходить не для всіх задач. Оцінка ефективності використання паралельних обчислень на GPU під час створення розробниками додаткових плагінів потребує знання факторів, які впливають на ефективність використання паралельних обчислень засобами графічної підсистеми. Визначення цих факторів може допомогти розробнику зрозуміти чи є взагалі сенс у реалізації паралельних обчислень на GPU конкретно для плагіну, що розроблюється.

2. МЕТОДИ ТА ТЕХНОЛОГІЇ ДОСЛІДЖЕННЯ

Реалізація програмного коду, що може використовувати архітектурні особливості GPU (неважливо, CUDA, OpenCL або Metal), передбачає формування двох програмних частин – kernel коду, який буде виконуватися на GPU та звичайного коду для центрального процесора. Задачу представляють у вигляді таблиці, розбитої на блоки, що в свою чергу складаються з потоків, які виконуються паралельно. Виконання kernel-коду на GPU забезпечує обробку кожним потоком одного елемента таблиці.

В CUDA Toolkit Documentation [7] наведено опис процесу обміну даними між робочим елементом (потоком) та CPU. З цього документу видно, що такий процес проходить з використанням глобальної пам'яті, яку сумісно використовують і GPU, і центральний процесор. З причини того, що швидкість обміну даними між CPU та GPU відносно низька, то, якщо з даними потрібно зробити невелику кількість операцій, у використанні GPGPU не має сенсу. Така ж ситуація виникає, якщо даних для обробки не дуже багато.

Зважаючи на те, що робота в кожному блоці виконується незалежно від інших, в невизначеному порядку, не існує засобів синхронізації між різними блоками, тобто засоби синхронізації робочих елементів присутні лише в межах одного блоку. Така обмеженість синхронізації обчислень значно звужує коло задач, що можуть використовувати для обчислень архітектуру GPU. Також відеокарта має обмеження на максимальний розмір робочої групи.

Врахування специфіки програмно-апаратної реалізації GPGPU по відношенню до практичних аспектів розробки плагінів для Adobe After Effects дозволяє відзначити два моменти:

1) Використання ресурсів графічного процесора виправдано лише для задач, де обчислення для одного пікселя не залежать від обчислень для іншого пікселя.

2) В коді потрібно потрібно уникати «ручного» обміну з буфером зображення між CPU та GPU, надаючи вказівник на область пам'яті графічного процесора де знаходиться оригінальне зображення, а також вказівник на область пам'яті для вихідного зображення.

Для аналізу ефективності використання паралельних обчислень на GPU у різних задачах, було створено декілька різних за принципом роботи плагінів:

– «Black and white», що робить зі звичайного кольорового зображення чорно-біле.

– «YUV controller», що представляє зображення у колірній моделі YUV і дозволяє змінювати значення її компонентів.

– «Gaussian blur», що реалізує розмивання Гауса.

– «Time slicer», що розділяє кадр на декілька частин.

Алгоритм роботи «Black and white» реалізує находження середнього значення складових RGB [8]. Код перетворення для одного пікселя представлено на рис. 1.

```
static PF_Err
pixel_calculations_8 (
    void       *refcon,
    A_long     xL,
    A_long     yL,
    PF_Pixel8  *inP,
    PF_Pixel8  *outP)
{
    PF_Err          err = PF_Err_NONE;

    double grayscale = inP->red * 0.333 + inP->green * 0.333 + inP->blue * 0.333;

    outP->red   = (A_u_char)grayscale;
    outP->green = (A_u_char)grayscale;
    outP->blue  = (A_u_char)grayscale;
    outP->alpha = inP->alpha;

    return err;
}
```

Рис. 1 – Обробка одного пікселя в плагіні «Black and white»

Плагін «YUV controller» представляє зображення у колірній моделі YUV і дозволяє змінювати значення її компонентів.

При конвертації RGB→YUV зважені значення R, G і B підсумовуються, щоб отримати Y (міру яскравості). U та V обчислюються як масштабовані відмінності між Y та значеннями B і R [9]. Реалізація алгоритму для обробки одиничного пікселя наведена на рис. 2.

```
static PF_Err
pixel_function_8(
    void        *refcon,
    A_long      xL,
    A_long      yL,
    PF_Pixel8   *inP,
    PF_Pixel8   *outP)
{
    PF_Err          err = PF_Err_NONE;

    YUVControllerParams *infoP = reinterpret_cast<YUVControllerParams *>(refcon);

    double red      = inP->red;
    double green    = inP->green;
    double blue     = inP->blue;

    double k_r = 0.299;
    double k_b = 0.114;

    // RGB -> YUV
    double y = k_r * red + (1 - k_r - k_b) * green + k_b * blue;
    double u = blue - y;
    double v = red - y;

    y = y * infoP->y / 100;
    u = u * infoP->u / 100;
    v = v * infoP->v / 100;

    // YUV -> RGB
    red     = y + v;
    green   = y - (k_r * v + k_b * u) / (1 - k_r - k_b);
    blue    = y + u;

    red     = min(max(red , 0),  PF_MAX_CHAN8);
    green   = min(max(green, 0), PF_MAX_CHAN8);
    blue    = min(max(blue, 0),  PF_MAX_CHAN8);

    outP->red   = (A_u_char)red;
    outP->green = (A_u_char)green;
    outP->blue  = (A_u_char)blue;
    outP->alpha = (A_u_char)inP->alpha;

    return err;
}
```

Рис. 2 – Обробка одного пікселя в плагіні «YUV controller»

Розмивання Гауса в плагіні «Gaussian blur» виконано стандартними методами [10]. Фрагмент коду, що ілюструє обробку по вертикалі в двовимірній матриці Гауса наведено на рис. 3.

В плагіні «Gaussian blur» на кожен піксель впливають 10 сусідніх пікселів з кожної сторони.

```
void
gaussian_blur_vertical(
    GaussianBlurInfoP   infoP,
    A_long      x,
    A_long      y)
{
    int radius = infoP->radius;

    double red_value    = 0;
    double green_value  = 0;
    double blue_value   = 0;
    double alpha_value  = 0;

    for (int kernelY = -radius; kernelY <= radius; kernelY++) {

        double kernelValue = infoP->kernel_weight_array[kernelY + radius];

        int currentY = y - kernelY;

        int position = (infoP->width * currentY + x) * 4;

        pixelData after_horizontal_blur_pixel_data = { 0, 0, 0, 0 };

        if (currentY > 0 && currentY < infoP->height) {
            after_horizontal_blur_pixel_data.red    = infoP->after_horizontal_blur_pixel_array[position];
            after_horizontal_blur_pixel_data.green  = infoP->after_horizontal_blur_pixel_array[position + 1];
            after_horizontal_blur_pixel_data.blue   = infoP->after_horizontal_blur_pixel_array[position + 2];
            after_horizontal_blur_pixel_data.alpha  = infoP->after_horizontal_blur_pixel_array[position + 3];
        }

        red_value   += after_horizontal_blur_pixel_data.red * kernelValue;
        green_value += after_horizontal_blur_pixel_data.green * kernelValue;
        blue_value  += after_horizontal_blur_pixel_data.blue * kernelValue;
        alpha_value += after_horizontal_blur_pixel_data.alpha * kernelValue;
    }

    int position = (y * infoP->width + x) * 4;

    infoP->after_vertical_blur_pixel_array[position]        = red_value;
    infoP->after_vertical_blur_pixel_array[position + 1]    = green_value;
    infoP->after_vertical_blur_pixel_array[position + 2]    = blue_value;
    infoP->after_vertical_blur_pixel_array[position + 3]    = alpha_value;
}
```

Рис. 3 – Обробка одного пікселя в плагіні «Gaussian blur»

Плагін «Time slicer» розділяє кадр на декілька частин. Центральна частина береться з поточного (оригінального) кадру. Частини справа – з наступних кадрів, а зліва – з вже минулих кадрів (рис. 4).

Плагін також дозволяє ділити зображення на декілька частин під нахилом і робити інтерполяцію для сусідніх частин.

Фрагмент коду, що ілюструє обробку пікселя в плагіні наведено на рис. 5.

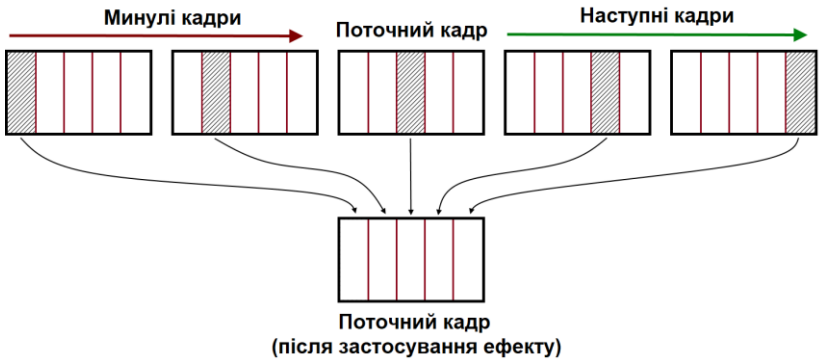

Рис. 4 – Принцип роботи плагіну «Time slicer»

```
int x = pixel_coordinates.x;
int y = pixel_coordinates.y;

double a = time_slice_info->line_data_1.a;
double b = time_slice_info->line_data_1.b;
double c = -a * x - b * y;

double c_1 = time_slice_info->line_data_1.c;
double c_2 = time_slice_info->line_data_2.c;

double c_min = min(c_1, c_2);
double c_max = max(c_1, c_2);
double c_middle = c_max - (c_max - c_min) / 2;
double c_current_distance_to_middle = max(c, c_middle) - min(c, c_middle);

double c_result_range_min = 0;
double c_result_range_max = 0;

double c_difference_total = c_max - c_min;
double c_half_difference_total = c_difference_total / 2;

if (need_interpolation) {
    double c_with_interpolation_range_min = c_min - c_half_difference_total;
    double c_with_interpolation_range_max = c_max + c_half_difference_total;

    c_result_range_min = c_with_interpolation_range_min;
    c_result_range_max = c_with_interpolation_range_max;
}
else {
    double c_without_interpolation_range_min = c_min;
    double c_without_interpolation_range_max = c_max;

    c_result_range_min = c_without_interpolation_range_min;
    c_result_range_max = c_without_interpolation_range_max;
}

if (c >= c_result_range_min && c <= c_result_range_max) {
    in_range = true;
}

if (in_range) {

    pixelData new_pixel_data = { 0, 0, 0, 0 };

    if (need_interpolation) {
```

Рис. 5 – Фрагмент коду обробки одного пікселя в плагіні «Time slicer»

Для порівняння ефективності використання паралельних обчислень у різних плагінах був вибраний наступний спосіб. До однохвилинного відео різної роздільної здатності застосовувався один із плагінів та вимірявся час рендерингу цього відео.

Виміри проводилися на процесорі AMD Ryzen 7 3750H і відеокарті NVIDIA GeForce 1650.

3. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Результати порівняння ефективності використання паралельних обчислень на GPU у різних плагінах дають наступні кількісні показники.

В таблиці 1 наведено час, затрачений на рендеринг відеофрагмента довжиною одна хвилина, засобами плагіну «Black and white». Для різних значень роздільної здатності робота плагіну виконувалася з використанням CUDA або ресурсів CPU.

Таблиця 1. Час рендерингу з використанням плагіну «Black and white»

Роздільна здатність відео, пікселі	**CPU, секунди**	**GPU, секунди**
1280x720	41	61
1920x1080	72	90
2560x1440	114	123
3840x2160	237	246

Для більшої зручності отримані дані представлено у вигляді графіків на рис. 6.

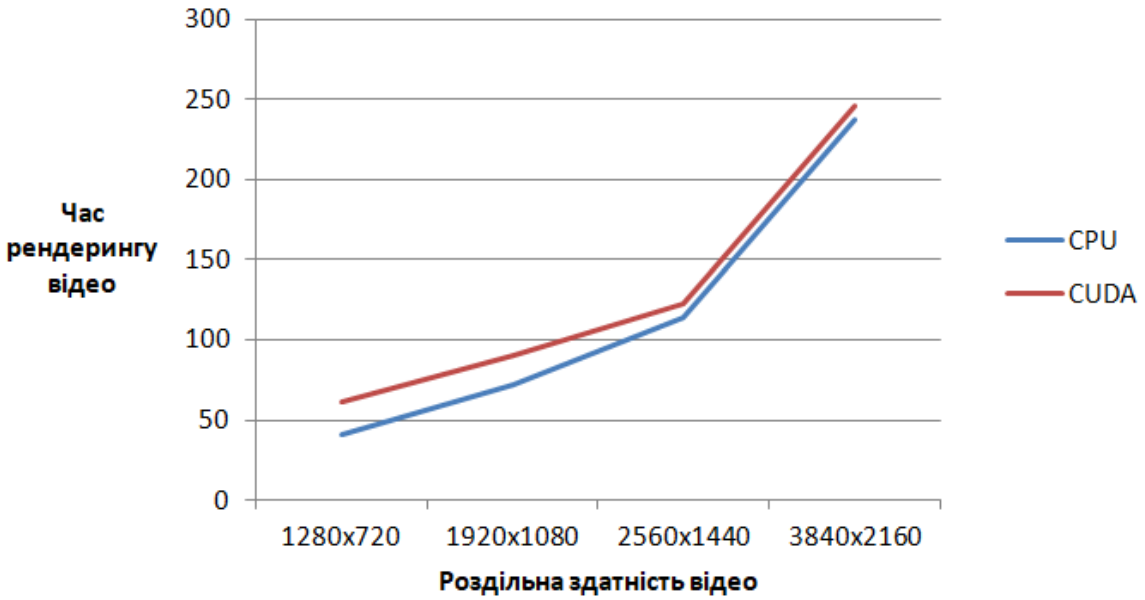

Рис. 6 – Ефективність використання паралельних обчислень на GPU в плагіні «Black and white»

Аналогічне повторення рендерингу для того самого фрагменту відео, але з використанням плагіну «YUV controller», дає результати, показані на рис. 7.

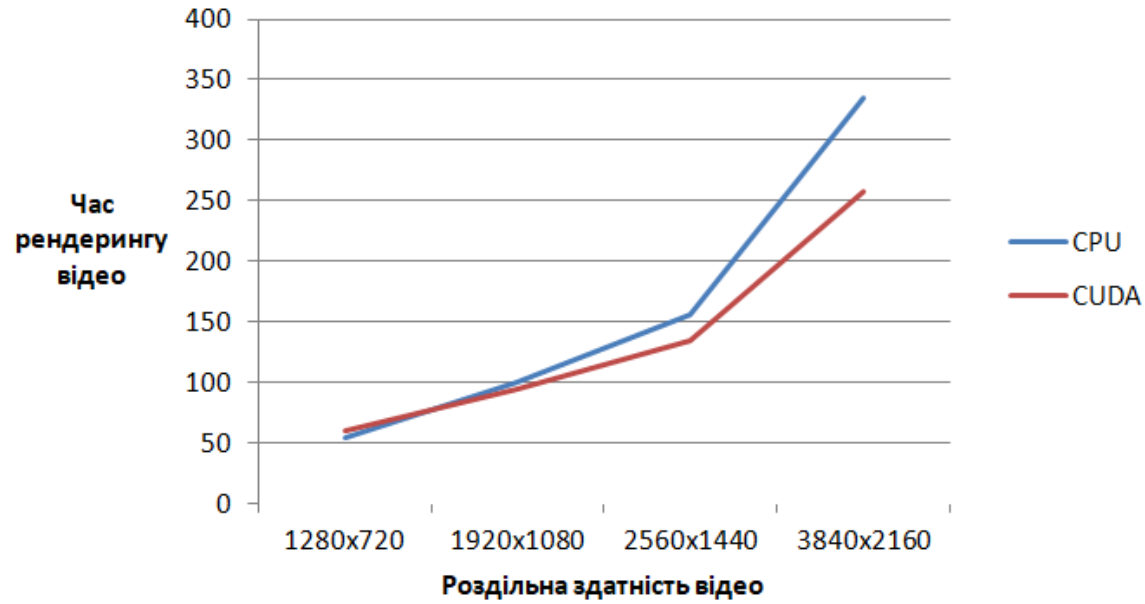

Рис. 7 – Ефективність використання паралельних обчислень на GPU в плагіні «YUV controller»

Час рендерингу 1-хвилинного відео при використанні плагіну «Gaussian blur» представлено на рис. 8.

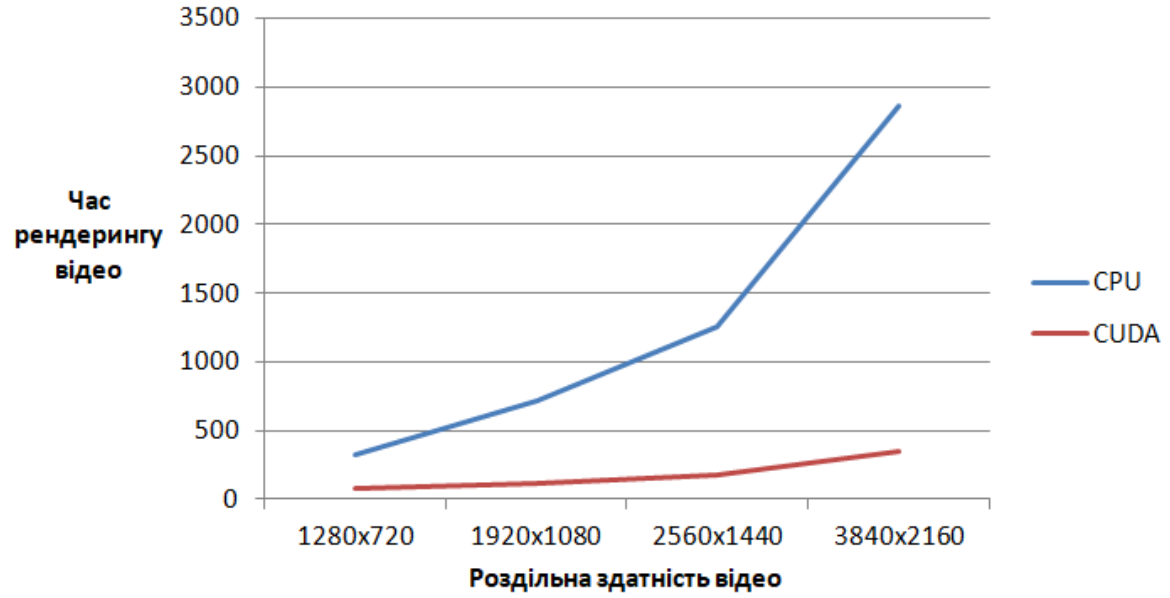

Рис. 8 – Ефективність використання паралельних обчислень на GPU в плагіні «Gaussian blur»

Ефективність використання паралельних обчислень у плагіні «Time slicer» залежить від його налаштувань (рис. 9).

Рис. 9 – Ефективність використання паралельних обчислень на GPU в плагіні «Time slicer»

Узагальнені результати досліджень ефективності використання паралельних обчислень з використанням можливостей графічного процесора для різних плагінів наведено в таблиці 2.

Таблиця 2. Порівняння ефективності використання паралельних обчислень на GPU для різних плагінів

	Black and white	**YUV controller**	**Gaussian blur**	**Time slicer**
Кількість арифметичних операцій для одного пікселя	5	≈25	≈730 (якщо враховувати горизонтальне і вертикальне розмивання)	(≈10÷25) (залежить від налаштувань плагіну і розташування пікселя)
Кількість пікселів, для яких необхідно виконати паралельні обчислення	Всі пікселі з поточного кадру	Всі пікселі з поточного кадру	Всі пікселі з поточного кадру	Всі пікселі з поточного та сусідніх кадрів
Ефективність використання паралельних обчислень на GPU	Неефективно	Іноді ефективно	Ефективно	Ефективно

Інтерпретація отриманих результатів дає змогу виділити декілька основних факторів, які впливають на ефективність використання паралелізму GPU-обчислень в плагінах Adobe After Effects:

- Паралелізація обчислень: можливість паралельного виконання основної частини обчислень для пікселів без синхронізації є ключовою. За потреби синхронізації, критично важливою є її мінімізація.
- Кількість пікселів: чим більше пікселів (включаючи сусідні кадри) потрібно обробити, тим ефективніше використання GPGU.
- Кількість обчислень на один піксель: велика кількість складних обчислень на піксель значно підвищує ефективність GPU.
- Роздільна здатність відео: вища роздільна здатність збільшує кількість пікселів, що робить GPU ефективнішим.
- Модель CPU та GPU: різні моделі мають різну продуктивність, тому в плагінах часто є вибір використання GPU.

Слід також враховувати фактори, які залишилися поза увагою дослідження, але, безсумнівно можуть також впливати на ефективність: технологія паралельних обчислень на GPU, версія After Effects, налаштування програми та проєкту, обсяг RAM, умовні переходи та тип операцій в коді обробки пікселя, оптимізація алгоритму під GPU, налаштування рендерингу, тощо.

IV. ВИСНОВКИ

Намагання розробників спеціалізованого ПЗ за замовчуванням використовувати можливості GPGU далеко не завжди мають сенс. Якщо не враховуються архітектурні особливості побудови графічного процесора та пов’язана з ними специфіка виконання обчислень, намагання підвищити ефективність обробки даних може мати зворотній ефект. Зокрема, розробникам плагінів для Adobe After Effects слід зважати на те, що використання паралельних обчислень на GPU дозволяє зменшити час обробки відео далеко не на всіх задачах.

Дослідження поведінки обчислювальної системи на реальних (не синтетичних) тестах дає змогу виділити всього три головних критерія оцінки потенціальної ефективності при розробці плагінів з використанням GPGU:

1) можливість виконання основної частини обчислень для пікселів паралельно;
2) загальну кількість пікселів кадру (групи кадрів), які потрібно обробити для знаходження кінцевого результату;
3) кількість необхідних обчислень для одного пікселя.

Таким чином, отримані результати дозволяють спробувати переглянути традиційний підхід до розробки спеціалізованих плагінів Adobe After Effects, що за замовчанням має на увазі використання технологій GPGU без уваги на доцільність ускладнення коду.

V. ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Враховуючи той факт, що Adobe After Effects підтримує не тільки CUDA, на якій в статті фактично й була зосереджена увага, можна припустити, що подібні дослідження інших технології використання GPGPU – OpenCL, OpenGL та Metal також можуть дати досить цікаві результати.

Робота в цьому напрямку може поступово нівелювати відношення щодо «неуніверсальності» графічних процесорів, з огляду на їхні архітектурні особливості, та сформувати більш раціональний підхід до написання коду, який би враховував програмно-апаратні особливості системи та специфіку алгоритму обробки пікселів зображення в конкретних задачах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] VIDEO COPILOT | *Element 3D V2 – 3D Object based Particle Plug-in* [Електронний ресурс] – Режим доступу: www. URL: https://www.videocopilot.net/products/element2/. Дата звернення: 03.03.2025.

[2] VIDEO COPILOT | After Effects Tutorials, Plug-ins and Stock Footage for Post Production Professionals [Електронний ресурс] – Режим доступу: www. URL: https://www.videocopilot.net/products/opticalflares/. Дата звернення: 03.03.2025.

[3] Red Giant – Magic Bullet | *Color Correction and Film Looks Plugin* [Електронний ресурс] – Режим доступу: www. URL: https://www.maxon.net/en/product-detail/red-giant/color/magic-bullet-looks. Дата звернення: 04.03.2025.

[4] Boris FX | *Continuum* [Електронний ресурс] – Режим доступу: www. URL: https://borisfx.com/products/continuum/?collection=continuum&product=continuum. Дата звернення: 04.03.2025.

[5] Boris FX | *Sapphire* [Електронний ресурс] – Режим доступу: www. URL: https://borisfx.com/products/sapphire/?collection=sapphire&product=sapphire. Дата звернення: 04.03.2025.

[6] GPU and GPU driver requirements for After Effects [Електронний ресурс] / Adobe. – Режим доступу: www. URL: https://helpx.adobe.com/ua/after-effects/using/basics-gpu-after-effects.html. Дата звернення: 14.03.2025.

[7] CUDA Toolkit Documentation | Programming Guide [Електронний ресурс] – Режим доступу: www. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Дата звернення: 20.03.2025.

[8] HPC/Кодинг/.NET/C++ CLI. Изменение цвета изображения из RGB в серые тона на C++/CLI. [Електронний ресурс] – Режим доступу: www. URL: https://s5.hpc.name/thread/u230/73011/izmenenie-cveta-izobrajeniya-iz-rgb-v-serye-tona-na-c-cli.html. Дата звернення: 25.03.2025.

[9] YUV, YCbCr, YPbPr color spaces [Електронний ресурс] – Режим доступу: www. URL: https://discovery-biz.net/enu0/faq/faq_YUV_YCbCr_YPbPr.html. Дата звернення: 31.03.2025.

[10] Gaussian blur [Електронний ресурс] – Режим доступу: www. URL: https://en.wikipedia.org/wiki/Gaussian_blur. Дата звернення: 04.04.2025.

Отримано 25.03.2023 р.

RESEARCH ON THE EFFICIENCY OF USING GPGU TECHNOLOGIES IN CUSTOM ADOBE AFTER EFFECTS PLUG-INS

Dmytro Ushenin,
Sergiy Sabanov,
Olha Shliaha

Zaporizhzhia Institute of Economics and Information Technologies
Zaporizhzhia, Ukraine

Abstract. The paper examines the features of using parallel computing technologies based on the computer’s graphics subsystem in Adobe After Effects plug-ins from third-party developers. The analysis showed that when creating plug-ins for the Adobe After Effects environment, developers quite often complicate the code, trying to transfer parallel computing from traditional means of the central processor to the functional blocks of the GPU, which at the present stage is considered absolutely correct. However, upon detailed examination, it turns out that the efficiency that the user receives from the operation of such plug-ins is by no means higher than the efficiency of plug-ins that have much simpler code that traditionally runs on the CPU. This situation is due to the lack of analysis of the features of the operation of parallel code in the graphics subsystem environment when processing various visual effects, and, as a consequence, the lack of clear recommendations on the necessity or, conversely, the inexpediency of working with the GPU hardware part in the code. This article presents a study of factors that affect the effectiveness of using parallel computing on GPUs in Adobe After Effects plugins. Recommendations are given on the feasibility of using GPUs depending on the specialization of the plugin related to the processing of a particular visual effect.

Keywords – *CPU, GPU, CUDA, OPENCL, Adobe After Effects, parallel computing, plugin*

Recieved 25.03.2023.