незалежна від мови програмування чи платформи, прикладного програмного інтерфейсу для візуалізації комп'ютерної графіки. Її підтримка дала змогу використовувати однаковий підхід для виводу графіки на різних платформах, а весь залежний код винести у окремий плагін. Для мобільних пристроїв при цьому використовується OpenGL ES — специфікація OpenGL для вбудованих систем.

Проект Lighthouse став основою нового етапу розвитку програмного каркасу Qt і передумовою для створення проекту Necessitas, метою якого стало створення порта Qt на системі Android, а також забезпечення простого методу керування, компілювання і розгортання додатку за допомогою Qt Creator. Успіх проекту Necessitas зумовив те, що з виходом Qt5 є доступна підтримка Qt на Android.

На сьогоднішньому етапі розробки (згідно огляду Qt5.3 alpha) Qt на Android уже забезпечує повний цикл розробки і розгортання додатку безпосередньо з Qt Creator на трьох основних платформах — Linux, Windows і Mac OS, а також має практично повну підтримку можливостей Qt, за виключенням хіба що повної підтримки модуля Qt WebKit. Крім цього, при встановленому наборі інструментів від розробників Android (Android SDK і Android NDK), Qt Creator версій 3.0.0 і вище забезпечує просте під'єднання і управління віртуальними Android-пристроями (Android Virtual Device, AVD), що значно прискорює процес розробки і тестування.

Проект Lighthouse сприяв ще одному значному кроку у напрямку підтримки Qt на мобільних платформах, а саме проекту Qt for iOS, який хоч і має значний успіх, все ж таки поступається у рівні розвитку Qt на Android. Проте уже зараз Qt for iOS підтримує значну частину можливостей Qt, використання коду Objective-C у Qt-додатках, що дає змогу зменшити проблеми зі створенням додатків для iOS, а також можливостями компілювання і розгортання програми за допомогою командного рядка і, при ряді додаткових налаштувань, безпосередньо з Qt Creator, для якого можна налаштувати підтримку мобільного пристрою. І хоч у порівнянні з можливостями розробки програм на Android, Qt for iOS виглядає більш скромніше, проте він все ж таки має значні можливості для створення кросплатформних програмних додатків.

Технологія CUDA — реалізація неграфічних обчислень на GPGPU

Парубочий В., Шувар Р.

Львівський національний університет імені Івана Франка, факультет електроніки, вул. Драгоманова 50, eddragonwolf@ukr.net

Technologies of GPGPU are an important and promising area of high-performance parallel computing. This paper deals with one of the most popular technologies today. It is CUDA. We considered the main aspects of architecture, principles of operation, and software (API, libraries and functions) provided for the development of parallel applications on graphics devices.

CUDA (Compute Unified Device Architecture - уніфікована обчислювальна архітектура пристроїв) - технологія неграфічних розрахунків на графічних процесорах (GPGPU, General-Purpose computation on GPUs), яка дає змогу реалізувати високопродуктивні паралельні обчислення на графічних процесорах завдяки структурі ядра мультипроцесора GPU.

Ефективність обчислень за допомогою CUDA досягається за рахунок структури графічного процесора, що містить велику кількість арифметико-логічних пристроїв, які за правильної організації обчислень можуть виконуватись швидше, ніж на центральних процесорах процесорах.

Технологія CUDA була анонсована компанією NVIDIA в 2006 році і підтримується на ряді відеокарт NVIDIA (серії відеокарт GeForce 8, GeForce 9, GeForce 200, Quadro і Tesla). Окрім апаратної підтримки CUDA вимагає спеціалізованого програмного забезпечення, що постачається вільно компанією NVIDIA у вигляді набору інструментів CUDA Toolkit [1], який містить компілятор nvcc, бібліотеки cuFFT, cuBLAS, cuRAND, cuSPARSE і NPP, профілювальник, налагоджувач gdb для GPU, CUDA runtime драйвер в комплекті стандартних драйверів NVIDIA, довідкову інформацію про техніку програмування на CUDA, а також CUDA Developer SDK, що містить вихідні коди, утиліти і документацію, необхідну для повноцінної розробки програм на CUDA.

Модель реалізації CUDA полягає у створенні абстракції низького рівня, яка дає змогу здійснювати зв'язок програми, що виконується на центральному процесорі, з графічним процесором і ініціалізувати на ньому обчислення. Ця абстракція забезпечується двома API: високорівневим CUDA Runtime API, що здійснює переведення усіх викликів реального часу у прості інструкції, що обробляються низькорівневим CUDA Driver API і передаються на графічний процесор для виконання. Хоча ці API і поділяються на високий і низький рівень, насправді вони є значно прив'язаними до апаратної реалізації технології і вимагають хорошого знання апаратної структури пристрою і принципів його роботи.

Куди більш високорівневими є бібліотеки, стандартні функції і ще одне API - CUDA Math API, що надаються технологією CUDA для простої розробки прикладних програм.

CUDA Math API надає стандартні математичні функції одинарної (single precision - float) і подвійної (double precision - double) точності, присутні у багатьох мовах програмування і адаптовані під архітектуру графічного процесора. Додаткову інформацію про API можна отримати за посиланням [2].

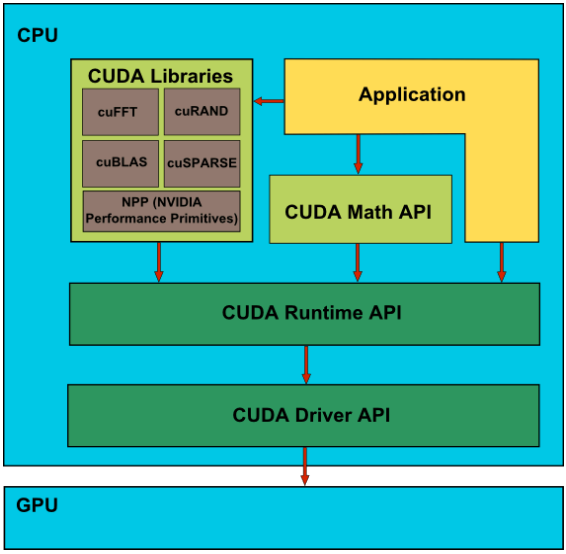

Рис. 1. Модель реалізації прикладної програми за допомогою технології CUDA.

Крім цього API CUDA надає також п'ять бібліотек для реалізації найбільш важливих операцій при обчисленні складних прикладних задач: cuFFT, cuBLAS, cuRAND, cuSPARSE і NPP.

cuFFT - CUDA-варіант бібліотеки швидкого перетворення Фур'є (FFTW), що широко використовується при аналізі сигналів, фільтрації і ряді інших задач обробки графічних і неграфічних даних. cuFFT надає простий інтерфейс для ефективного обчислення швидкого перетворення Фур'є і підтримує одно-, дво- і тривимірне перетворення комплексних і дійсних даних, пакетне виконання для декількох одновимірних трансформацій в паралелі. Для двовимірних і тривимірних трансформацій підтримуються розміри даних в межах [2, 16384], а для одновимірної трансформації - до 8 мільйонів елементів [3].

cuBLAS надає стандартні алгоритми лінійної алгебри. Бібліотеку дуже легко використовувати, необхідно створити матрицю і векторні об'єкти в пам'яті відеокарти, заповнити їх даними, викликати необхідні функції cuBLAS, і завантажити результати з відеопам'яті назад в системну пам'ять. cuBLAS містить спеціальні функції для створення і знищення об'єктів в пам'яті GPU, а також для читання і запису даних у цю пам'ять. Серед стандартних функцій бібліотеки BLAS бібліотека cuBLAS підтримує перший (векторно-векторні операції), другий (векторно-матричні операції) і третій (матрично-матричні операції) рівні для дійсних чисел і перший рівень CGEMM для комплексних [4].

cuRAND надає методи для простої і ефективної генерації псевдовипадкових і квазівипадкових чисел. Псевдовипадкова

послідовність чисел, згенерована запропонованим алгоритмом, задовольняє більшість статистичних властивостей випадкових послідовностей, проте генерується за допомогою детермінованого алгоритму, а квазівипадкова послідовність n-вимірних точок породжується детермінованим алгоритмом, що максимально рівномірно заповнює n-вимірний простір. Бібліотека складається з двох частин - бібліотеки, що працює на базі центрального процесора, і заголовкового файлу для графічного процесора. Бібліотека нічим не відрізняється від інших бібліотек, що працюють на базі центрального процесора. Для її використання необхідно під'єднати заголовковий файл /include/curand.h, щоб отримати оголошення функцій, а потім зв'язатися з бібліотекою. Випадкові числа можуть бути згенеровані як для графічного пристрою, так і в пам'яті центрального процесора. Для отримання випадкових чисел в пам'яті графічного пристрою необхідно викликати бібліотеку на хості, але генерація випадкових чисел здійснюватиметься на графічному процесорі і згенеровані значення зберігатимуться в пам'яті графічного пристрою. Після чого згенеровані значення можна використовувати на пристрої, або скопіювати у глобальну пам'ять обчислювальної машини для подальшої обробки на центральному процесорі. У випадку генерації на хості обчислення виконуються на центральному процесорі, а усі згенеровані значення в оперативній пам'яті обчислювальної машини. Друга частина бібліотеки - заголовковий файл /include/curand_kernel.h - визначає функції графічного пристрою для створення станів генератора випадкових чисел і генерації випадкових послідовностей. Код користувача може використовувати цей заголовковий файл для безпосереднього виклику функцій ядра і генерування випадкових чисел в локальній пам'яті, без необхідності запису і читання з глобальної пам'яті [5].

Бібліотека cuSPARSE містить набір базових підпрограм лінійної алгебри, що використовуються для обробки розріджених матриць.

Підпрограми бібліотеки можна розділити на чотири категорії:

- Рівень 1: операції між векторами в розрідженій і в щільній формі;
- Рівень 2: операції між матрицями в розрідженій і в щільній формі;
- Рівень 3: операції між матрицею в розрідженій формі і набором векторів у щільній формі (який також може розглядатися як матриця у щільному формі);
- Конверсія: операції, що забезпечують перетворення між різними формами матриць.

cuSPARSE дає змогу розробникам отримати доступ до обчислювальних ресурсів графічного процесора проте не підтримує розпаралелювання на декілька GPU [6].

NVIDIA NPP - це найбільш високорівнева бібліотека, що використовує обчислювальні процеси CUDA. Функціональність бібліотеки орієнтована на візуалізацію і обробку відео і широко використовується у цих областях а також для представлення результатів неграфічних обчислень [7].

Оскільки графічний і центральний процесори не пов'язані безпосередньо один з одним і не мають спільного адресного простору, обмін даними між глобальною пам'яттю обчислювальної машини і пам'яттю графічного пристрою можливий лише за допомогою копіювання даних між цими типами пам'яті. Це досягається за допомогою методу cudaMemcpy():

cudaError_t cudaMemcpy(void* dst, const void* src, size_t count, cudaMemcpyKind kind)

тут dst - вказівник на ділянку пам'яті, в яку буде скопійовано дані, src - вказівник на дані, що будуть скопійовані, count - розмір даних, kind - вид (напрямок) копіювання.

Останній параметр є списком cudaMemcpyKind, який містить наступні напрямки для копіювання даних:

cudaMemcpyHostToHost (0) - дані копіюються з ділянки глобальної пам'яті обчислювальної машини у іншу ділянку її пам'яті;

cudaMemcpyHostToDevice (1) - дані копіюються з ділянки глобальної пам'яті обчислювальної машини у пам'ять графічного пристрою;

cudaMemcpyDeviceToHost (2) - дані копіюються з ділянки пам'яті графічного пристрою в глобальну пам'ять обчислювальної машини;

cudaMemcpyDeviceToDevice (3) - дані копіюються з ділянки пам'яті графічного пристрою в іншу ділянку пам'яті пристрою;

cudaMemcpyDefault (4) - дані копіюються у уніфікований віртуальний адресний простір.

Виділення пам'яті на графічному пристрої також потребує особливого підходу. Для цього використовується метод cudaMalloc():

cudaError_t cudaMalloc(void** devPtr, size_t size),

де devPtr - вказівник на виділену пам'ять, а size - розмір виділеної пам'яті на графічному пристрої.

Після закінчення обчислень на графічному процесорі може виникнути потреба у очищенні пам'яті графічного пристрою. Це можна здійснити за допомогою методу cudaFree():

cudaError_t cudaFree(void* devPtr),

де devPtr - вказівник на ділянку пам'яті, яку необхідно очистити.

Крім того, можна скористатися методом cudaDeviceReset(), який очищає усі ресурси і стани графічного пристрою і перезапускає його:

cudaError_t cudaDeviceReset (void).

Розглянутих функцій цілком достатньо для реалізації двовимірного Фур'є-перетворення з копіюванням даних з пам'яті обчислювальної машини, трансформації на графічному пристрої і копіюванні результатів назад у пам'ять обчислювальної машини:

```
//Оголошення заголовкового файлу бібліотеки cuFFT
#include <cufft.h>

float *hostData;
... //Ініціалізація даних у пам'яті обчислювальної машини (вектор
    hostData розміру size = width * height)
```

```
//Ініціаліція даних в пам'яті графічного пристрою
float *deviceData;
//Виділення пам'яті розміром size
cudaMalloc((void**)&deviceData, sizeof(float) * size);

//Копіювання даних з пам'яті обчислюваної машини в пам'ять
    графічного пристрою
cudaMemcpy(deviceData, hostData, sizeof(float) * size,
    cudaMemcpyHostToDevice);

//Виділення пам'яті для результату Фур'є-перетворення
cufftComplex *outData;
cudaMalloc((void**)&outData, sizeof(cufftComplex) * size);

//Ініціалізація плану двовимірного дійсно-комплексного Фур'є-
    перетворення
cufftHandle fftPlan;
cufftPlan2d(&fftPlan, width, height, CUFFT_R2C);

//Виконання плану
cufftExecR2C(fftPlan, (cufftReal*)deviceData, outData);

//Виділення ділянки пам'яті обчислювальної машини для результатів
    Фур'є-перетворення
cufftComplex *outHostData;
outHostData = (cufftComplex *)malloc(sizeof(cufftComplex) * size);

//Копіювання даних з пам'яті графічного пристрою в пам'ять
    обчислювальної машини
cudaMemcpy(outHostData, outData, sizeof(cufftComplex) * size,

    cudaMemcpyDeviceToHost);

//Очищення пам'яті графічного пристрою
cufftDestroy(fftPlan);
cudaFree(deviceData);
cudaFree(outData);
//Очищення пам'яті обчислювальної машини
free(hostData);
free(outHostData);
//Перезапуск графічного пристрою
cudaDeviceReset();
```

Простий програмний інтерфейс CUDA і високі показники швидкодії і паралелізму програм, реалізованих за допомогою цієї технології, є визначальними чинниками популярності і актуальності використання технології CUDA для реалізації неграфічних обчислень не лише для реалізації паралельних обчислень на персональних комп'ютерах, а й на високопродуктивних обчислювальних кластерах гібридної архітектури. Крім того, графічний процесор з підтримкою технології CUDA може використовуватися як додатковий обчислювальний пристрій для вирішення на центральному процесорі обчислювальної машини трудомістких задач, в яких можна виділити блоки, що можуть виконуватися паралельно на графічному процесорі, не завантажуючи при

цьому центральний процесор, що може бути використаний для обчислень послідовного типу.

Література

1. https://developer.nvidia.com/cuda-toolkit — сторінка завантаження набору інструментів CUDA Toolkit.
2. http://docs.nvidia.com/cuda/cuda-math-api/index.html#axzz2z8JwGKOm — довідкова інформація про можливості CUDA Math API.
3. http://docs.nvidia.com/cuda/cufft/index.html#axzz2z8JwGKOm — документація бібліотеки cuFFT.
4. http://docs.nvidia.com/cuda/cublas/index.html#axzz2z8JwGKOm — сторінка документації бібліотеки cuBLAS.
5. http://docs.nvidia.com/cuda/curand/index.html#axzz2z8JwGKOm — довідкова інформація бібліотеки cuRAND.
6. http://docs.nvidia.com/cuda/cusparse/index.html#axzz2z8JwGKOm — довідка бібліотеки cuSPARSE.
7. http://docs.nvidia.com/cuda/npp/index.html#axzz2z8JwGKOm — сторінка для завантаження документації бібліотеки NPP.

Контроль за віддаленим обладнанням та керування на прикладі використання послідовного порту (RS-232)

Петрів М.М.

Івано-Франківський національний технічний університет нафти і газу
hifi.ua@gmail.com

Низька якість каналів зв’язку стимулює до пошуку альтернативних каналів доступу та методів зменшення об’ємів переданої та прийнятої інформації. А необхідність у низькорівневому контролі змушує до задіяння не тільки програмних, але й апаратних засобів. У свою чергу використання Open Source забезпечує велику гнучкість конфігурацій, безпеку та зменшення вартості, а перевірені часом апаратні рішення дають змогу забезпечити надійну роботу в цілому. Таким чином Open Source підхід вирішує цілу низку поставлених вимог як в освітніх цілях, так і в Enterprise-сфері.

Станом на 2014 рік доступність виділених швидкісних каналів у мережу Інтернет в Україні у більшості великих міст не складає жодних проблем. Проте, за їх межами єдиними доступними каналами залишаються GSM і CDMA мережі мобільних операторів. На даний момент швидкість роботи цих мереж, якість каналу та час відгуку залишаються незадовільними. Якщо в таких умовах роботи системному адміністратору або оператору потрібно провести налаштування або контроль, або навпаки в даному місці знаходиться віддалене обладнання, то стандартні графічні методи доступу через VNC або RDP втрачають актуальність. У випадку ж користувачів GNU/Linux є можливість доступу по SSH, що дає змогу в умовах низькошвидкісних каналів працювати в більш-менш стабільному режимі. Проте, якщо ми глянемо на мережеву