УДК 004.4'24 https://doi.org/10.15407/pp2019.01.027

А.Ю. Дорошенко, М.М. Бондаренко, О.А. Яценко

АВТОМАТИЗОВАНЕ ПРОЕКТУВАННЯ OPENCL ПРОГРАМ НА ОСНОВІ АЛГЕБРО-АЛГОРИТМІЧНОГО ПІДХОДУ

Подальший прогрес у покращенні показників якості створення паралельних програм пов'язаний з використанням гетерогенних архітектур обчислювальних систем. Неоднорідні паралельні системи включають у себе, зокрема, гібридні обчислювальні платформи, що поєднують використання центральних процесорів і графічних прискорювачів. Одним з інструментальних засобів програмування таких систем є OpenCL. У статті виконане налаштування раніше створеного алгебро-алгоритмічного інструментарію проектування і синтезу на автоматизовану разробку OpenCL-програм. Особливістю запропонованого підходу до проектування є використання мови, що ґрунтується на системах алгоритмічних алгебр Глушкова. Підхід продемонстровано на проектуванні програми інтерполяції для задачі метеорологічного прогнозування. Проведено експеримент з виконання згенерованої за допомогою розробленого інструментарію паралельної програми на графічному прискорювачі. Виконане порівняння з реалізацією програми на CUDA.

Ключові слова: автоматизоване проектування програм, алгебра алгоритмів, метеорологічне прогнозування, неоднорідні паралельні обчислювальні системи, синтез програм, CUDA, OpenCL.

Вступ

Паралельні та розподілені обчислення на мультипроцесорних платформах на даний час є основним джерелом забезпечення необхідних потреб у високій продуктивності обчислень при розв’язанні складних науково-технічних і господарських проблем. Подальший прогрес у покращенні показників якості створення паралельних програм пов’язаний із використанням неоднорідних архітектур обчислювальних систем. Неоднорідні паралельні системи включають у себе, зокрема, гібридні обчислювальні платформи, що поєднують використання центральних процесорів (Central Processing Units, CPUs) та графічних прискорювачів (Graphics Processing Units, GPUs). Одним з інструментальних засобів програмування таких систем є OpenCL (Open Computing Language) [1] – фреймворк для створення програмних застосувань, що виконуються у гетерогенному середовищі. На відміну від Nvidia CUDA [2], яка є реалізацією GPGPU (обчислення загального призначення на графічних процесорах) лише для відеокарт Nvidia, OpenCL є специфікацією, яку можуть реалізовувати різні виробники апаратного забезпечення. Існують реалізації OpenCL для відеокарт Nvidia, AMD, Intel, ARM, а також програмованих користувачем вентильних матриць (FPGA). Також на відміну від платформи Nvidia CUDA, програми для якої потрібно компілювати за допомогою спеціального компілятора, програми OpenCL можна збирати будь-яким компілятором, треба лише вказати бінарний файл бібліотеки при збірці. Зазначимо, що програмування гібридних систем – це досить складна задача порівняно з розробкою застосувань для однорідних архітектур, тому актуальним є питання розробки спеціальних засобів автоматизації розробки програмного забезпечення, що дозволяли б найбільш ефективно генерувати найпродуктивніший код для таких систем.

В роботах [3, 4] запропоновані теорія, методологія та інструментарій для автоматизованого проектування, синтезу та перетворення послідовних та паралельних програм, що ґрунтуються на засобах алгебр алгоритмів та переписувальних правил. В роботах [5–7] розроблені засоби застосовані для генерації паралельних програм для платформи Nvidia CUDA на основі схем алгоритмів. У даній роботі виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез програм, що використовують OpenCL. Застосування інструментарію продемонстроване на розробці паралельної програми інтерполяції, що

входить до складу програми чисельного прогнозування погоди [6, 7]. Проведено експеримент з виконання згенерованої паралельної OpenCL програми на графічному прискорювачі.

Запропонований у даній статті підхід є близьким до робіт, присвячених генерації OpenCL програм [8–11]. Зокрема, в роботі [8] розглядається підхід та програмний засіб Gaspard2 для специфікації, проектування та генерації вищезгаданих програм на основі використання уніфікованої мови моделювання UML. В роботі [9] запропоновані програмний засіб STEPOCL та предметно-орієнтована мова для спрощення розробки програм, що використовують декілька прискорювачів одночасно. В [10] розглядається автоматичний генератор функцій-ядер OpenCL для задач лінійної алгебри на основі високорівневих специфікацій, поданих користувачем у предметно-орієнтованій мові, що ґрунтується на C++. В ргототі [11] запропонований підхід до генерації OpenCL на основі високорівневих функціональних специфікацій та використання переписувальних правил.

Відмінність підходу, викладеного у даній статті, полягає у використанні для автоматизованого проектування паралельних програм специфікацій систем алгоритмічних алгебр Глушкова, поданих у природно-лінгвістичній формі, що полегшує розуміння алгоритмів і досягнення необхідної якості програм. Іншою перевагою розроблених інструментальних засобів є застосування методу діалогового конструювання синтаксично правильних програм, що виключає можливість виникнення синтаксичних помилок у процесі проектування схем.

1. Алгебро-алгоритмічні засоби проектування та їх налаштування на розробку OpenCL програм

В основу пропонованого підходу до проектування паралельних програм покладений апарат систем алгоритмічних алгебр (САА) та їх модифікацій [3, 4]. Модифіковані САА (САА-М) призначені для формалізації процесів мультиобробки, що виникають при конструюванні програмного забезпечення в мультипроцесорних системах. На САА-М ґрунтуються розроблені інструментальні засоби автоматизованого проектування та генерації програм [3–7].

1.1. Системи алгоритмічних алгебр. Модифіковані САА є двоосновною алгеброю $< Pr, Op; \Omega >$, де Pr – множина логічних умов (предикатів); Op – множина операторів; Ω – сигнатура, що складається з логічних операцій (диз'юнкції, кон'юнкції, заперечення, лівого множення оператора на умову) та операторних операцій (композиції, альтернативи, циклу та ін.), що будуть розглянуті далі. Предикати та оператори можуть бути базисними або складеними. Базисні елементи вважаються атомарними, неподільними абстракціями та пов'язані з предметною областю алгоритму, що проектується. Складені оператори будуються з елементарних за допомогою операцій послідовного й паралельного виконання операторів.

На САА-М ґрунтується алгоритмічна мова САА/1 [5, 6], призначена для багаторівневого структурного проектування і документування послідовних та паралельних алгоритмів і програм. Перевагою її використання є можливість опису алгоритмів у природно-лінгвістичній формі. Подання операторів мовою САА/1 називаються САА-схемами.

Далі наведено перелік назв та специфікації основних операторних операцій сигнатури САА-М, поданих у природньо-лінгвістичній формі.

1. Композиція (послідовне виконання операторів):

"*operator* 1";
"*operator* 2".

2. Альтернатива (умовний оператор):

IF '*condition*' THEN
  "*operator* 1"
ELSE
  "*operator* 2"
END IF.

3. Цикл типу while:

WHILE ‘*condition*’
LOOP “*operator*”
END OF LOOP.

4. Цикл типу for:

FOR (*counter* FROM *start* TO *fin*)
LOOP “*operator*”
END OF LOOP.

5. Синхронізатор, що виконує затримку обчислень доти, поки значення умови не стане істинним:

WAIT ‘*condition*’.

В роботі [5] розглядаються додаткові операції, призначені для формалізованого проектування програм, що використовують платформу Nvidia CUDA. В роботі [6] згадані засоби використані для проектування програми чисельного прогнозування погоди.

1.2. Розширення САА-М операціями, орієнтованими на паралельні обчислення в гетерогенному середовищі. В даному підрозділі розглядаються нові операції алгебри алгоритмів, призначені для проектування OpenCL програм [1]. OpenCL є технологією створення програмних застосувань, що виконуються у гетерогенному середовищі. OpenCL поєднує прикладний програмний інтерфейс (API) та варіант мови С для програмування та одночасного використання різних пристроїв, здатних виконувати паралельні обчислення (наприклад, CPU, GPU та Xeon Phi). Для координації роботи пристроїв у гетерогенному середовищі присутній один “головний” пристрій (host), який взаємодіє з усіма іншими обчислювальними пристроями (device) за допомогою OpenCL API. OpenCL програма працює з так званими платформами (platform). Платформа є програмним пакетом, який надається відповідним розробником апаратних засобів (наприклад, AMD Accelerated Parallel Processing, Nvidia CUDA, Intel OpenCL). Платформа OpenCL складається з головного пристрою, з’єднаного з обчислювальними пристроями, що підтримують OpenCL. Кожний OpenCL-пристрій складається з обчислювальних блоків (compute unit), які далі розділяються на один або більше елементів-обробників (processing elements, PE). Програма в OpenCL складається з двох частин: головної, яка виконується на керуючому пристрої, та обчислювальної – однієї або кількох функцій-ядер, що виконуються на елементах-обробниках OpenCL-пристроїв. Головна частина програми визначає середовище (контекст), у якому виконуються ядра, і керує їх виконанням. Контекст включає платформу, обчислювальні пристрої та буфери пам’яті для них. Для обчислювального пристрою в рамках контексту створюється черга команд для виконання (command queue). Операції з обчислювальним пристроєм, такі, як читання/запис даних і запуск ядра, заносяться в цю чергу і послідовно виконуються. Коли ядро ставиться в чергу на виконання, визначається простір індексів. Копія ядра виконується для кожного індексу з цього простору і називається робочою одиницею (задачею, work-item). Робочі одиниці організовані в групи (work-groups) і виконуються паралельно на елементі-обробнику обчислювального пристрою.

Далі наведено послідовність дій для створення OpenCL програм та відповідні нові базисні оператори, додані в алгебру алгоритмів.

1. Отримання списку доступних платформ та запис їх у змінну *pl*:

“Get all available platforms (*pl*)”.

2. Отримання списку пристроїв з отриманої платформи *pl*:

“Get all devices (*dvs*) available on a platform (*pl*)”.

3. Створення контексту виконання для пристроїв:

“Create a context (*cnt*) for devices (*dvs*)”.

4. Створення черги виконання для пристрою:

"Create a command queue (*cmdqueue*)
for a context (*cnt*) and a device (*dv*)".

5. Компіляція файлу, що містить вихідний код функції-ядра OpenCL:

"Create a kernel (*krnl*) from
a source (*program*)",

де *krnl* – назва функції-ядра; *program* – шлях до файлу з вихідним кодом цієї функції.

Наведемо приклад САА-схеми для функції-ядра, що виконує додавання двох векторів *a* та *b* довжини *n*:

SCHEME VECTOR ADD ====

"vecAdd(a, b, c, n)"
==== "Declare a variable (id) of type (int)";
(id := "Get the global work-item
identifier for dimension (0)");
IF (id < n)
THEN
"(c[id]) := (a[id] + b[id])"
END IF;

END OF SCHEME VECTOR ADD

У схемі базисний оператор "Get the global work-item identifier for dimension (0)" повертає глобальний ідентифікатор задачі.

6. Створення буферу для даних, зазначених у змінній *var*:

"Create a memory buffer (*buff*)
for data (*var*) on devices in
the context (*cnt*)".

7. Завантаження буферу для змінної *var* в пам'ять пристрою за допомогою черги:

"Add commands to queue (*cmdqueue*)
writing a buffer of data (*var*) from
host to device".

8. Установка значення *arg_value* для параметра під номером *arg_index* ядра *krnl*:

"Set the argument value (*arg_value*)
for a parameter (*arg_index*) of
a kernel (*krnl*)".

9. Завантаження ядра *krnl* в чергу пристрою та його асинхронне виконання:

"Add a command to queue
(*cmdqueue*) executing a kernel
(*krnl*)(*globalworksize*)
(*localworksize*) on a device",

де *globalworksize* – загальна розмірність простору індексів; *localworksize* – розмірність локальної підмножини задач у групі.

10. Синхронізація (очікування завершення) виконання усіх задач, що були занесені в чергу:

WAIT 'All previously queued
commands in (*cmdqueue*)
are issued to the device and
have completed'.

11. Зчитування даних з буферу, в який записувався результат:

"Add commands to queue
(*cmdqueue*) reading from a buffer
of data (*var*) from device to host".

Приклад використання вищенаведених конструкцій розглядається у розділі 2.

1.3. Інструментарій автоматизованої розробки програм. Розроблений інтегрований інструментарій проектування та синтезу програм (ІПС) ґрунтується на використанні розглянутих засобів САА-М та методу діалогового конструювання синтаксично правильних програм (ДСП-методу) [3, 4]. На відміну від традиційних синтаксичних аналізаторів, ДСП-метод орієнтований не на пошук і виправлення синтаксичних помилок, а на виключення можливості їх появи в процесі побудови алгоритму. Основна ідея методу полягає у порівневому конструюванні схем зверху

вниз шляхом суперпозиції мовних конструкцій САА-М. На кожному кроці конструювання система надає користувачу на вибір лише ті конструкції, підстановка яких у текст алгоритму, що проектується, не порушує синтаксичну правильність схеми. На основі побудованої схеми алгоритму виконується автоматична генерація тексту програми цільовою мовою програмування. Відображення операцій САА-М у текст мовою програмування подане у вигляді шаблонів і зберігається в базі даних інструментарію.

Основними компонентами системи ІПС є такі:

- діалоговий конструктор синтаксично правильних програм (ДСП-конструктор), призначений для діалогового проектування схем алгоритмів та синтезу програм мовами Java та C, C++;
- редактор граф-схем;
- база даних алгеброалгоритмічних специфікацій, у якій зберігається текст конструкцій САА-М і базисних елементів схем, а також їх програмні реалізації;
- генератор САА-схем за гіперсхемами.

Для автоматизації виконання трансформацій алгоритмів система ІПС може застосовуватися спільно з системою TermWare [12, 13], що ґрунтується на парадигмі переписувальних правил. В роботі [7] на основі використання TermWare розроблено програмний засіб для оптимізації обчислень, що дозволяє в напівавтоматичному режимі здійснювати паралелізацію циклічних операторів програм, що використовують Nvidia CUDA.

З метою налаштування системи ІПС на проектування і генерацію OpenCL програм, в базу даних системи включені нові конструкції, розглянуті вище в підрозділі 1.2.

2. Приклад застосування системи ІПС для проектування OpenCL програми

У даному розділі розглядається використання інтегрованого інструментарію для автоматизованої розробки паралельної OpenCL програми інтерполяції початкових метеорологічних даних, що входить до складу програмного забезпечення чисельного прогнозування погоди [6, 7].

Далі наведено фрагмент початкової послідовної САА-схеми інтерполяції, побудованої в [7].

```
SCHEME INTERPOLATION
SEQUENTIAL

“interpolation_sequential”
==== FOR (h FROM 0 TO Pk–1)
       LOOP
         FOR (k FROM 0 TO Lmz–1)
         LOOP
           FOR (j FROM 0 TO Mmz–1)
           LOOP
             FOR (i FROM 0 TO Nmz–1)
             LOOP
               “(a) := ((WZZ + US[h][k][j][i] /
                0.321f) * Rs * VS[h][k][j][i])”;
               “(Tp) := (TS[h][k][j][i] *
                pow(1000.f / HS[h][k][j][i],
                2.f/7.f))”;
               “(Tv) := (Tp * (1.f + 0.6078f *
                QS[h][k][j][i]))”;
               “(Qc[h][k][j][i]) := (a – (0.5f *
                Tv + (1.f – Zmz[k]) * g *
                F_X[j][i] / 0.321f))”
             END OF LOOP
           END OF LOOP
         END OF LOOP
       END OF LOOP;

END OF SCHEME
INTERPOLATION SEQUENTIAL
```

Наведена схема є чотирма вкладеними циклами за ітераторами $h \in [0 \ldots$ $\ldots Pk-1]$, $k \in [0 \ldots Lmz-1]$, $j \in [0 \ldots Mmz-$ $-1]$, $i \in [0 \ldots Nmz-1]$, де Pk, Lmz, Mmz, Nmz – натуральні числа, що визначають розмір скінченно-різницевої сітки. У схемі US, VS, TS, HS, QS – вхідні масиви метеорологічних величин (горизонтальні складові вектору швидкості вітру, температура, вологість і т. п.); Qc – вихідний масив. Оскільки ітерації у циклах є незалежними, вони можуть бути виконані паралельно.

Проектування паралельної OpenCL програми за допомогою системи ІПС включає у себе розробку САА-схем для

функції-ядра та керуючої програми, а також генерацію на їх основі відповідного коду мовою програмування.

Далі наведена САА-схема для функції-ядра, сконструйована за допомогою інструментарію ІПС на основі перетворення послідовної схеми. У параметрах, вказаних у дужках після назви функції (interpolation) зазначені вхідні та вихідні дані для задачі. Розпаралелювання здійснюється за індексами h, k, j (їх значення обчислюється за допомогою оператора "Get the global work-item identifier for dimension"). За індексом i в ядрі виконується цикл основних обчислень.

```
SCHEME INTERPOLATION KERNEL

"interpolation(US, VS, HS, QS, TS, F_x,
Zmz, Qc, Pk, Lmz, Mmz, Nmz)"
==== "Declare the list of variables (h, k, j) of
        type (int)";
      (h := "Get the global work-item
            identifier for dimension (0)");
      (k := "Get the global work-item
            identifier for dimension (1)");
      (j := "Get the global work-item
            identifier for dimension (2)");
      IF NOT((h >= Pk) OR (k >= Lmz) OR
           (j >= Mmz))
      THEN
        FOR (i FROM 0 TO Nmz–1)
        LOOP
          "Declare a constant (ind) of type
           (unsigned int)";
          "Declare the list of variables
           (a, Tp, Tv) of type (float)";
          "(ind) := (h + Pk * k + Pk * Lmz *
            j + Pk * Lmz * Mmz * i)";
          "(a) := ((WZZ + US[ind] / 0.321f)
            * Rs * VS[ind])";
          "(Tp) := (TS[ind] *
           pow(1000.f / HS[ind], 2.f / 7.f))";
          "(Tv) := (Tp * (1.f + 0.6078f *
                  QS[ind]))";
          "(Qc[ind]) := (a – (0.5f * Tv +
                       (1.f – Zmz[k]) * g *
                       F_x[j + i * Mmz] /
                                  0.321f))"
        END OF LOOP
      END IF;

END OF SCHEME INTERPOLATION
KERNEL
```

На відміну від реалізації алгоритму для CUDA [7], в якій дані US, VS, HS, QS, TS, Qc зберігалися як чотиривимірні вказівники, що є неефективним і не дозволяє використовувати кешування даних процесором, в реалізації алгоритму на OpenCL, що розглядається у даній роботі, дані передаються суцільним блоком (continuous array). Індекс для обробки поточного елемента масиву обчислюється спеціальним чином та заноситься у змінну ind. При розпаралелюванні послідовного алгоритму звертання до елементів згаданих масивів (наприклад, US[h][k][j][i]) замінюється на відповідний запис з індексом ind (наприклад, US[ind]).

Далі наведена САА-схема головної частини програми, побудована на основі кроків 1–11, розглянутих у підрозділі 1.2.

```
SCHEME INTERPOLATION HOST

"main(argc, argv)"
==== "Read input parameters
      (Pk, Lmz, Mmz, Nmz)
      from command line (argc, argv)";
      "Get all available platforms (pl)";
      "Get all devices (dvs) available on a
       platform (pl[0])";
      "Create a context (cnt) for
       devices (dvs)";
      "Create a command queue
       (cmdqueue) for a context (cnt) and a
       device (dvs[0])";
      "Create a kernel ("interpolation")
       from a source
       ("kernels/interpolation.cl")";
      "Declare and fill 4D continuous arrays
       (US, VS, HS, QS, TS, Qc)
       of type (auto) and size
       (Pk, Lmz, Mmz, Nmz)";
      "Declare and fill a 2D continuous
```

```
    array (F_X) of type (auto)
    and size (Mmz, Nmz)";
  "Declare and fill a 1D continuous
    array (Zmz) of type (auto) and
    size (Nmz)";
  "Create memory buffers for data
    (US, VS, HS, QS, TS, Qc, F_X,Zmz)
    on devices in the context (cnt)";
  "Add commands to queue (cmdqueue)
    writing buffers of data
    (US, VS, HS, QS, TS, F_X, Zmz)
    from host to device";
  "Set the argument values for
    parameters of a kernel (krnl)";
  "Add a command to queue
    (cmdqueue) executing a kernel
    (krnl)(3)({Pk, Lmz, Mmz})
    on a device";
  WAIT 'All previously queued
    commands in (cmdqueue) are issued
    to the device and have completed';
  "Add commands to queue (cmdqueue)
    reading from a buffer of data (Qc)
    from device to host";

END OF SCHEME INTERPOLATION
HOST
```

На основі побудованих САА-схем в системі ІПС виконана генерація програмного коду OpenCL, результати виконання якого на графічному прискорювачі наведено у наступному розділі.

3. Результати експерименту

Проведено експеримент з виконання розробленої паралельної програми в обчислювальному середовищі, що складається з двоядерного процесора Intel Core i5-4210U (частота 1,7–2,7 ГГц) та графічного процесора nVidia GeForce 840M (384 ядра, 2 Гб оперативної пам'яті). Для порівняння виконано також відповідну програму, що використовує Nvidia CUDA [7].

На рис. 1 показано графік залежності часу виконання програми інтерполяції від розміру вхідних даних на CPU, а також програм CUDA та OpenCL. На рис. 2 показано графіки залежності мультипроцесорного прискорення

$$Sp = \frac{T_{CPU}}{T_{GPU}},$$

де T_{CPU}, T_{GPU} — час виконання на CPU та GPU, відповідно.

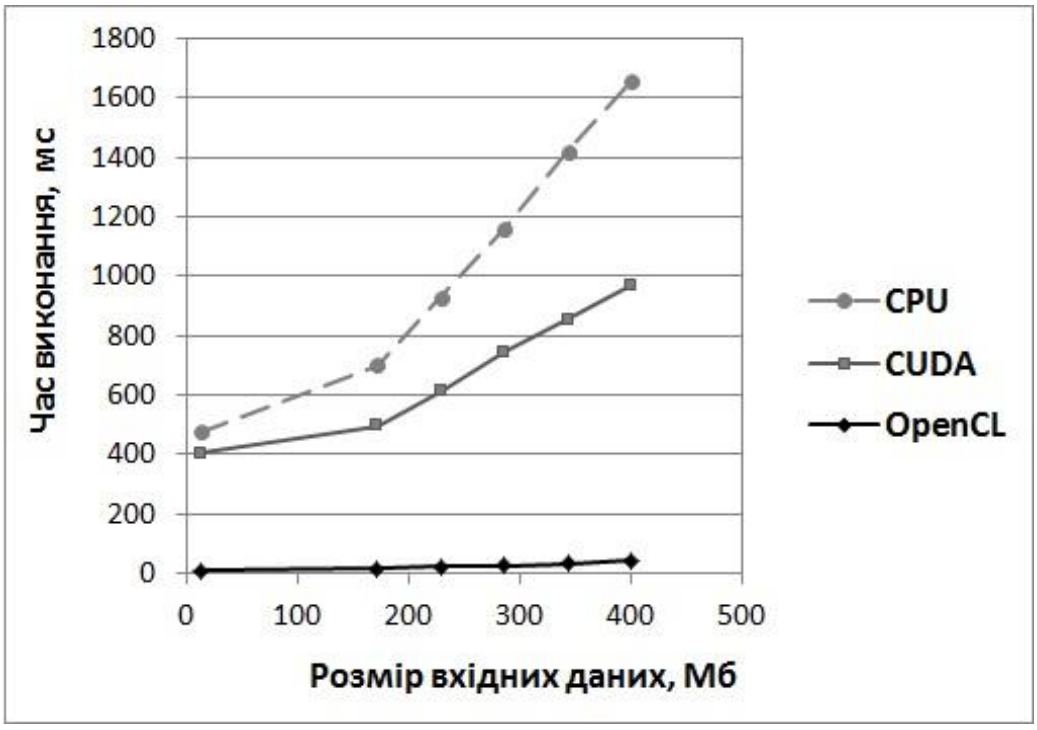

Рис. 1. Залежність часу виконання програми інтерполяції на CPU, CUDA та OpenCL від розміру вхідних даних

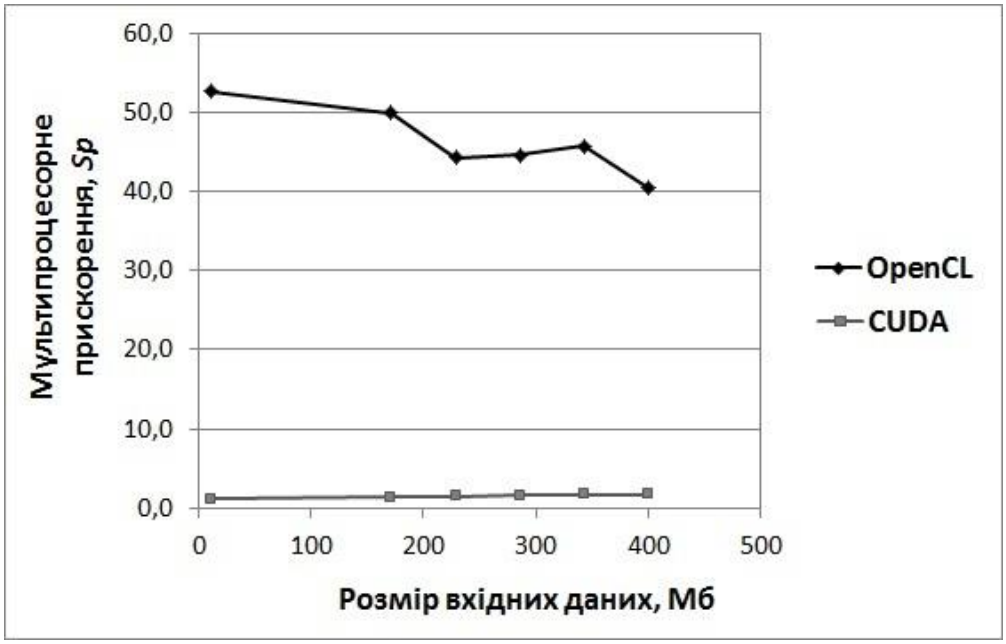

Рис. 2. Залежність мультипроцесорного прискорення від розміру вхідних даних для реалізацій паралельної програми інтерполяції на OpenCL та CUDA

Як видно з наведених результатів, OpenCL програма значно випереджає відповідний варіант для CUDA. Максимальне значення прискорення *Sp* OpenCL програми становить 52,7 (за розміру вхідних даних 11,451 Мб), для CUDA програми максимальне значення *Sp* є 1,7 (за розміру вхідних даних 400,553 Мб). Підвищена швидкодія OpenCL програми порівняно з реалізацією для CUDA пояснюється використанням одновимірних масивів даних замість чотиривимірних (див. розділ 2).

Висновки

Виконане налаштування алгебро-алгоритмічного інструментарію на формалізоване проектування та синтез OpenCL програм. Особливістю запропонованого підходу до проектування є використання мови САА-схем, перевагою якої є простота в навчанні і використанні, а також застосування методу конструювання синтаксично правильних схем, який виключає можливість появи синтаксичних помилок у процесі проектування алгоритмів.

Проведено експеримент з виконання згенерованої за допомогою системи ІПС паралельної програми інтерполяції початкових даних для задачі метеорологічного прогнозування на графічному прискорювачі. Результати експерименту продемонстрували гарний ступінь розпаралелюваності обчислень та переваги використання OpenCL порівняно з Nvidia CUDA для згаданої задачі.

Література

1. OpenCL overview. The open standard for parallel programming of heterogeneous systems [Електронний ресурс]. Режим доступу до ресурсу: https://www.khronos.org/opencl (дата звернення 22.01.2019 р.).
2. Nvidia CUDA technology [Електронний ресурс]. Режим доступу до ресурсу: http://www.nvidia.com/cuda (дата звернення 22.01.2019 р.).
3. Андон Ф.И., Дорошенко А.Е., Жереб К.А., Шевченко Р.С., Яценко Е.А. Методы алгебраического программирования. Формальные методы разработки параллельных программ. Киев: Наукова думка, 2017. 440 с.
4. Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А. Алгеброалгоритмические модели и методы параллельного программирования. Киев: Академпериодика, 2007. 631 с.
5. Дорошенко А.Ю., Бекетов О.Г., Жереб К.А., Яценко О.А. Формалізоване проектування та синтез паралельних програм для

відеографічних прискорювачів. *Проблеми програмування.* 2013. № 3. С. 38–46.
6. Дорошенко А.Ю., Бекетов О.Г., Прусов В.А., Тирчак Ю.М., Яценко О.А. Формалізоване проектування та генерація паралельної програми чисельного моделювання погоди. *Проблеми програмування.* 2014. № 2–3. С. 72–81.
7. Дорошенко А.Ю., Яценко О.А., Бекетов О.Г. Алгоритм автоматизованого розпаралелювання циклічних операторів для графічних прискорювачів. *Проблеми програмування.* 2017. № 4. С. 28–36.
8. Rodrigues A., Guyomarc'h F., Dekeyser J.-L. An MDE approach for automatic code generation from UML/MARTE to OpenCL. *Computing in Science and Engineering*. 2012. Vol. 15, N 1. P. 46–55.
9. Li P., Brunet E., Trahay F., Parrot C., Thomas G., Namyst, R. Automatic OpenCL code generation for multi-device heterogeneous architectures. *Proc. 44th International Conference on Parallel Processing* (ICPP 2015), Beijing, China (1–4 September, 2015). Piscataway, NJ: IEEE, 2015. P. 959–968.
10. Tillet P., Rupp K., Selberherr S. An automatic OpenCL compute kernel generator for basic linear algebra operations. *Proc. 2012 Symposium on High Performance Computing* (HPC'12), Orlando, Florida, USA (26–30 March, 2012). San Diego, CA: Society for Computer Simulation International, 2012. P. 4:1–4:2.
11. Steuwer M., Fensch C., Lindley S., Dubach C. Generating performance portable code using rewrite rules: from high-level functional expressions to high-performance OpenCL code. *Proc. 20th ACM SIGPLAN International Conference on Functional Programming* (ICFP'15), Vancouver, Canada (31 August – 2 September, 2015). New York: ACM, 2015. P. 205–217.
12. Doroshenko A., Shevchenko R. A rewriting framework for rule-based programming dynamic applications. *Fundamenta Informaticae*. 2006. Vol. 72, N 1–3. P. 95–108.
13. Doroshenko A., Zhereb K., Yatsenko O. Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. *Communications in Computer and Information Science. Information and Communication Technologies in Education, Research, and Industrial Applications*. Springer, Heidelberg, 2013. Vol. 412. P. 70–92.

References

1. OpenCL Overview. The open standard for parallel programming of heterogeneous systems. [Online] Available from: https://www.khronos.org/opencl [Accessed: 22 January 2019]
2. Nvidia CUDA technology. [Online] Available from: http://www.nvidia.com/cuda [Accessed: 22 January 2019]
3. Andon, P.I. et al. (2017) Methods of algebraic programming. Formal methods of parallel program development. Kyiv: Naukova dumka. (in Russian)
4. Andon P.I. et al. (2007) Algebra-algorithmic models and methods of parallel programming. Kyiv: Akademperiodyka. (in Russian)
5. Doroshenko A.Yu., Beketov O.G., Zhereb K.A. & Yatsenko O.A. (2013) Formalized designing and synthesis of parallel programs for graphics processing units. Problems in programming. (3). P. 38–46. (in Ukrainian)
6. Doroshenko A.Yu., Beketov O.G., Prusov V.A., Tyrchak Yu.M. & Yatsenko O.A. (2014) Formalized designing and generation of parallel program for numerical weather forecasting task. *Problems in programming.* (2–3). P. 72–81. (in Ukrainian)
7. Doroshenko A.Yu., Yatsenko O.A. & Beketov O.G. (2017) Algorithm for automatic loop parallelization for graphics processing units. *Problems in programming*. (4). P. 28–36. (in Ukrainian)
8. Rodrigues A., Guyomarc'h F. & Dekeyser J.-L. (2012) An MDE approach for automatic code generation from UML/MARTE to OpenCL. Computing in Science and Engineering. 15 (1). P. 46–55.
9. Li P., Brunet E., Trahay F., Parrot C., Thomas G. & Namyst R. (2015) Automatic OpenCL code generation for multi-device heterogeneous architectures. In Proc. 44th International Conference on Parallel Processing (ICPP 2015). Beijing, China, 1–4 September 2015. Piscataway, NJ: IEEE. P. 959–968.
10. Tillet P., Rupp K. & Selberherr S. (2012) An automatic OpenCL compute kernel generator for basic linear algebra operations. In Proc. 2012 Symposium on High Performance Computing (HPC'12). Orlando, Florida, USA, 26-30 March 2012. San Diego, CA: Society for Computer Simulation International. P. 4:1–4:2.

11. Steuwer M., Fensch C., Lindley S. & Dubach C. (2015) Generating performance portable code using rewrite rules: from high-level functional expressions to high-performance OpenCL code. In Proc. 20th ACM SIGPLAN International Conference on Functional Programming (ICFP'15). Vancouver, Canada, 31 August – 2 September 2015. New York: ACM. P. 205–217.
12. Doroshenko A. & Shevchenko R. (2006) A rewriting framework for rule-based programming dynamic applications. Fundamenta Informaticae. 72 (1–3). P. 95–108.
13. Doroshenko A., Zhereb K. & Yatsenko O. (2013) Developing and optimizing parallel programs with algebra-algorithmic and term rewriting tools. In Proc. 9th International Conference "ICT in Education, Research, and Industrial Applications" (ICTERI 2013), Revised Selected Papers. Kherson, Ukraine, 19-22 June 2013. Berlin: Springer. 412. P. 70–92.

Одержано 10.02.2019

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу
теорії комп'ютерних обчислень,
професор кафедри автоматики та
управління в технічних системах
НТУУ "КПІ імені Ігоря Сікорського ".
Кількість наукових публікацій в
українських виданнях – понад 150.
Кількість наукових публікацій в
зарубіжних виданнях – понад 50.
Індекс Хірша – 5.
http://orcid.org/0000-0002-8435-1451,

Бондаренко Микола Миколайович,
магістр факультету комп'ютерних наук
та кібернетики.
http://orcid.org/0000-0002-6362-414X,

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник.
Кількість наукових публікацій в
українських виданнях – 43.
Кількість наукових публікацій в
зарубіжних виданнях – 13.
Індекс Хірша – 2.
http://orcid.org/0000-0002-4700-6704.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, м. Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 3559.

Київський національний університет
імені Тараса Шевченка,
01601, Київ, вул. Володимирська, 60.

E-mail: doroshenkoanatoliy2@gmail.com,
bondarenko_mykola@yahoo.com.ua,
oayat@ukr.net