УДК 681.3

А.Ю. Дорошенко, О.Г. Бекетов, Р.Б. Іванів, В.О. Іовчев, І.О. Мироненко, О.А. Яценко

АВТОМАТИЗОВАНА ГЕНЕРАЦІЯ ПАРАЛЕЛЬНИХ ПРОГРАМ ДЛЯ ГРАФІЧНИХ ПРИСКОРЮВАЧІВ НА ОСНОВІ СХЕМ АЛГОРИТМІВ

Запропоновано алгеброалгоритмічний підхід до автоматизованого проектування та генерації програм для графічних прискорювачів. Особливість підходу полягає у використанні високорівневих специфікацій, поданих в алгебрі алгоритмів, а також застосуванні методу, який забезпечує синтаксичну правильність алгоритмів та програм, що проектуються. Підхід реалізовано в онлайновому інструментарії, призначеному для діалогового конструювання схем алгоритмів та генерації програм у цільових мовах програмування. Застосування інструментарію проілюстровано на прикладі розробки паралельної програми для задачі конвективної дифузії.

Вступ

Подальше поширення багатопотокового паралелізму в сучасних обчислювальних системах привело до потреби створення спеціального інструментарію для розробки паралельного програмного забезпечення, який охоплює усі етапи життєвого циклу програми. У відділі теорії комп'ютерних обчислень Інституту програмних систем НАН України упродовж тривалого періоду розвивається теорія, методологія та інструментарій для автоматизованого проектування паралельних програм, що ґрунтуються на засобах високорівневої алгеброалгоритмічної формалізації і автоматизації перетворень програм [1–5]. Зокрема, розроблено Інтегрований інструментарій Проектування та Синтезу програм (ІПС) [2, 3] та його нову версію – Онлайновий Діалоговий конструктор Синтаксично Правильних програм (ОДСП). Автоматизована система ОДСП [4, 5] є онлайновим сервісом, що використовує технологію Web 2.0, і призначений для проектування та генерації програм на основі високорівневих специфікацій алгоритмів. Специфікації алгоритмів (схеми) ґрунтуються на використанні алгебри алгоритмів Глушкова [1]. Система забезпечує побудову послідовних та паралельних алгоритмів у режимі порівневого конструювання, та генерацію відповідних програм в цільових мовах програмування Java, C++. У попередніх роботах інструментарій ОДСП застосований для розробки паралельних програм для багатоядерних центральних процесорів (central processing units, CPUs). Мета даної роботи полягає в налаштуванні системи ОДСП на розробку паралельних програм для графічних прискорювачів (graphics processing units, GPUs), які дозволяють отримати значне підвищення продуктивності обчислень порівняно з CPU. Розглядається проблема автоматизованого проектування та генерації паралельних програм для NVIDIA CUDA [6], яка є однією з обчислювальних платформ та програмних моделей для GPU.

Існуючі підходи до генерації коду для GPU архітектур ґрунтуються, зокрема, на анотаціях для опису властивостей структур даних та областей коду [7], мережах процесів Кана [8], директивах компілятора [9], графах потоку даних [10], високорівневих абстракціях структур даних, задач та комунікаційних операторів [11].

Відмінністю підходу, запропонованого у даній роботі, є використання алгебраїчних специфікацій, поданих у природно-лінгвістичній формі, яка полегшує розуміння алгоритмів та досягнення необхідної якості програм. Інша перевага розроблених засобів полягає у застосуванні методу діалогового конструювання синтаксично правильних програм, який виключає можливість виникнення синтаксичних помилок у процесі проектування схем.

Матеріал даної роботи організований таким чином. У розділі 1 розглянуто операції алгебри алгоритмів, що викорис-

товуються для проектування паралельних програм для графічних прискорювачів. У розділі 2 розглянуто архітектуру та основні функції системи ОДСП. У розділі 3 проілюстровано застосування розроблених алгебро-алгоритмічних засобів на прикладі проектування паралельної програми для розв'язання задачі конвективної дифузії [12, 13]. Роботу завершують висновки та напрямки подальшої роботи.

1. Алгебра алгоритмів та формалізоване проектування паралельних програм для графічних прискорювачів

В основу пропонованого підходу до проектування паралельних програм покладений апарат систем алгоритмічних алгебр (САА) Глушкова [1], що призначені для формалізованого проектування послідовних і паралельних алгоритмів та програм. САА є двоосновною алгеброю $<\{U, B\}; \Omega>$, де U – множина операторів, B – множина логічних умов; Ω – сигнатура, що складається з предикатних (кон'юнкція, диз'юнкція, заперечення) та операторних (композиція, альтернатива, цикл) операцій. Подання алгоритмів у САА (алгебраїчні формули) називаються регулярними схемами. Окрім алгебраїчної форми, алгоритми можуть бути подані в природно-лінгвістичній (САА-схеми) та графовій (граф-схеми) формах.

В розробленій авторами системі ОДСП [4, 5], яка розглядається у розділі 2 даної статті, застосовується природно-лінгвістична форма проектування, що ґрунтується на мові САА/1 [1]. Основними об'єктами згаданої мови є абстракції операторів і умов, що можуть бути елементарними (базисними) або складеними. Складені оператори й умови будуються з елементарних за допомогою операцій послідовного і паралельного виконання операторів. В ОДСП алгоритми проектуються англійською мовою.

Назви та текст основних САА операцій є такими:

- композиція (послідовне виконання) операторів:

```
operator1; operator2
```

- альтернатива (оператор розгалуження):

```
If (predicate) Then
   operator1
Else
   operator2
End If
```

- оператор циклу:

```
While loop (predicate)
   operator
End of loop
```

З метою орієнтації САА на проектування програм для графічних прискорювачів NVIDIA, що підтримують технологію CUDA, в сигнатуру Ω включені додаткові конструкції. Відмітимо, що апаратно графічні прискорювачі NVIDIA складаються з деякої кількості CUDA-ядер, кожне з яких здатне одночасно виконувати певну кількість потоків (threads). Ядром (kernel) називається підпрограма (функція) для GPU, що виконується потоками. Модель програмування в CUDA передбачає групування потоків у блоки. Кожен блок є масивом потоків, які взаємодіють між собою за допомогою спільної пам'яті і точок синхронізації. Блоки, у свою чергу, об'єднуються в сітку блоків.

Додаткові конструкції САА для проектування програм для GPU є такими:

- визначення функції-ядра:

```
KERNEL fname(param_list) =
   function_body;
```

де `fname` – назва функції; `param_list` – список формальних параметрів; `function_body` – реалізація функції, представлена в САА;

- операція виклику функції-ядра:

```
fname(Nb, Nth, arg_list),
```

де `fname` – назва функції; `Nb` – кількість блоків у сітці; `Nth` – кількість потоків у кожному блоці; `arg_list` – список фактичних параметрів функції;

- оператори, що отримують значення глобального (унікального) індексу потоку та індекс потоку в межах блоку

(локальний індекс), і присвоюють отримане значення цілочисельній змінній i:

```
Get global index of
the thread (i);

Get local index of
the thread (i);
```

• синхронізатор – операція, що здійснює очікування завершення обчислень усіма потоками:

```
Synchronizer (all threads
completed work);
```

• операції виділення та звільнення відеопам'яті для певної змінної:

```
Allocate the memory for
variable on GPU (var_name,
count);

Free the memory for varia-
ble on GPU (var_name);
```

де var_name – назва змінної; count – розмір пам'яті, яку необхідно виділити, в байтах;

• операції копіювання даних із пам'яті CPU у відеопам'ять і у зворотному напрямку:

```
Copy data from CPU to GPU
(dest, src, count);

Copy data from GPU to CPU
(dest, src, count);
```

де dest – назва змінної, яка є призначенням копіювання; src – назва змінної, значення якої необхідно скопіювати; count – розмір даних, що копіюються, в байтах.

Приклад. Проілюструємо використання деяких з вищенаведених операцій для побудови алгоритму для GPU, призначеного для паралельного множення елементів одновимірного масиву на певне число. CAA-схема даного алгоритму є такою.

```
SCHEME Multiplication of array
elements by a value, for GPU;

KERNEL Multiply array elements
       by value (A, N, val) =
   Get global index of
   the thread (i);
   Assign value
   (A[i], A[i] * val);


Main function =
   Declare array (h_A, int, N);
   Declare array (d_A, int);
   Initialize array with
   random values (h_A, N);
   Print array (h_A, N);
   Allocate the memory for
   variable on GPU (d_A,
   A_size);
   Copy data from CPU to GPU
   (d_A, h_A, A_size);
   Multiply array elements
   by value (Nb, Nth, d_A, N, 20);
   Copy data from GPU to CPU
   (h_A, d_A, A_size);
   Free the memory for
   variable on GPU (d_A);
   Print array (h_A, N);
```

Схема складається з реалізації функції-ядра та основної функції схеми. Функція-ядро, яка має назву " Multiply array elements by value (A, N, val) ", виконує множення елемента A[i] одновимірного цілочисельного масиву A на число val; тут N – довжина масиву, i – індекс потоку. В реалізації основної функції (Main) виконується оголошення масиву h_A, що підлягає обробці і зберігається у пам'яті CPU, а також ініціалізація цього масиву довільними значеннями. Наведено також оголошення та виділення пам'яті для відповідного масиву d_A у пам'яті GPU. Далі виконується копіювання значень масиву h_A в масив d_A. Після цього здійснюється виклик вищезгаданої функції-ядра, в результаті якого виконується паралельне множення елементів масиву d_A на значення val = 20. Значення параметрів Nb та Nth (кількість блоків у сітці та кількість потоків у кожному блоці) обираються таким чином, що Nb * Nth = N. Далі копіюються дані із масиву d_A в масив h_A. Після цього виконується звіль-

нення зарезервованої відеопам'яті та виведення значень результуючого масиву `h_A` на екран.

Застосування операцій САА для проектування паралельного алгоритму для розв'язання задачі конвективної дифузії буде розглянуте у розділі 3.

2. Онлайновий діалоговий конструктор синтаксично правильних програм

Розроблена автоматизована система ОДСП призначена для діалогового проектування та генерації програм [4, 5]. Система ґрунтується на використанні систем алгоритмічних алгебр (див. розділ 1) та методу діалогового конструювання синтаксично правильних програм (ДСП-методу) [1]. На відміну від традиційних синтаксичних аналізаторів, ДСП-метод орієнтовано не на пошук та виправлення синтаксичних помилок, а на виключення можливості їх появи у процесі побудови алгоритмів. Основна ідея методу полягає у порівневому конструюванні схем зверху вниз шляхом суперпозиції мовних конструкцій САА. На кожному кроці конструювання система в діалозі з користувачем надає на вибір лише ті конструкції, підстановка яких у текст, що формується, не порушує синтаксичну правильність схеми.

Особливість системи ОДСП (порівняно з раніше розробленим авторами інструментарієм ІПС [2, 3]) полягає в орієнтації на багатокористувальницьке використання інструментарію через Інтернет і розподілену архітектуру системи. Відмітимо, що компоненти інструментарію є автономними, гнучко зв'язаними і мають узгоджений протокол обміну даними. Таким чином, дана система, по суті, є сервісно-орієнтованою.

Інструментарій ОДСП складається з компонентів, показаних на рис. 1.

Клієнт є Web-інтерфейсом для діалогової взаємодії користувача з інструментарієм та його ресурсами. Він надає можливість виконувати конструювання схеми алгоритму із використанням елементів бази даних, здійснювати генерацію коду цільовою мовою програмування (Java або C++) та виконувати запуск програми в обчислювальному середовищі. В основу проектування алгоритмів у системі покладено діалоговий режим з використанням списку конструкцій та дерева алгоритму (див. рис. 2). Специфікації конструкцій ґрунтуються на англомовній версії мови САА/1 (див. розділ 1). Побудова схем здійснюється зверху вниз за допомогою деталізації конструкцій. У процесі проектування схеми користувач за чергою обирає необхідні

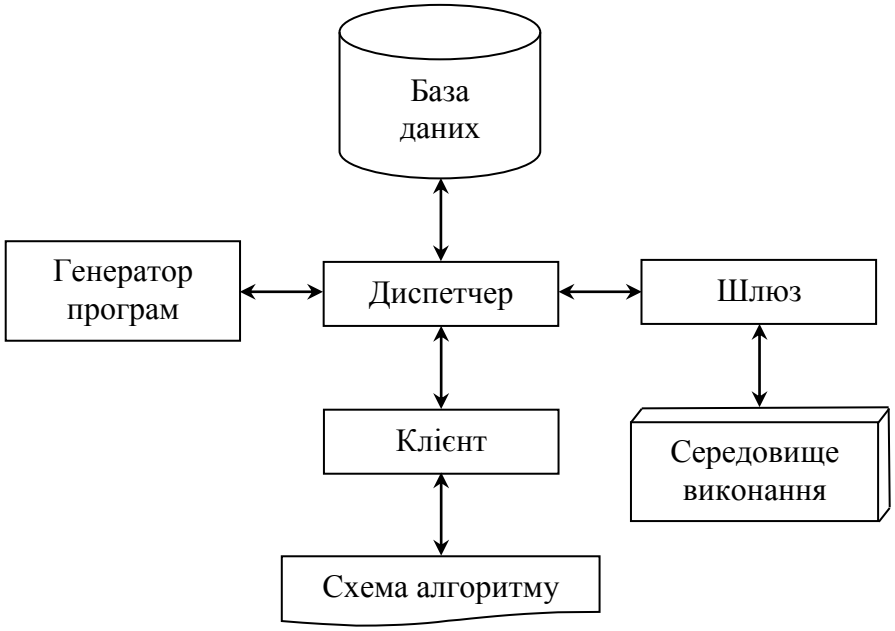

Рис. 1. Архітектура системи ОДСП

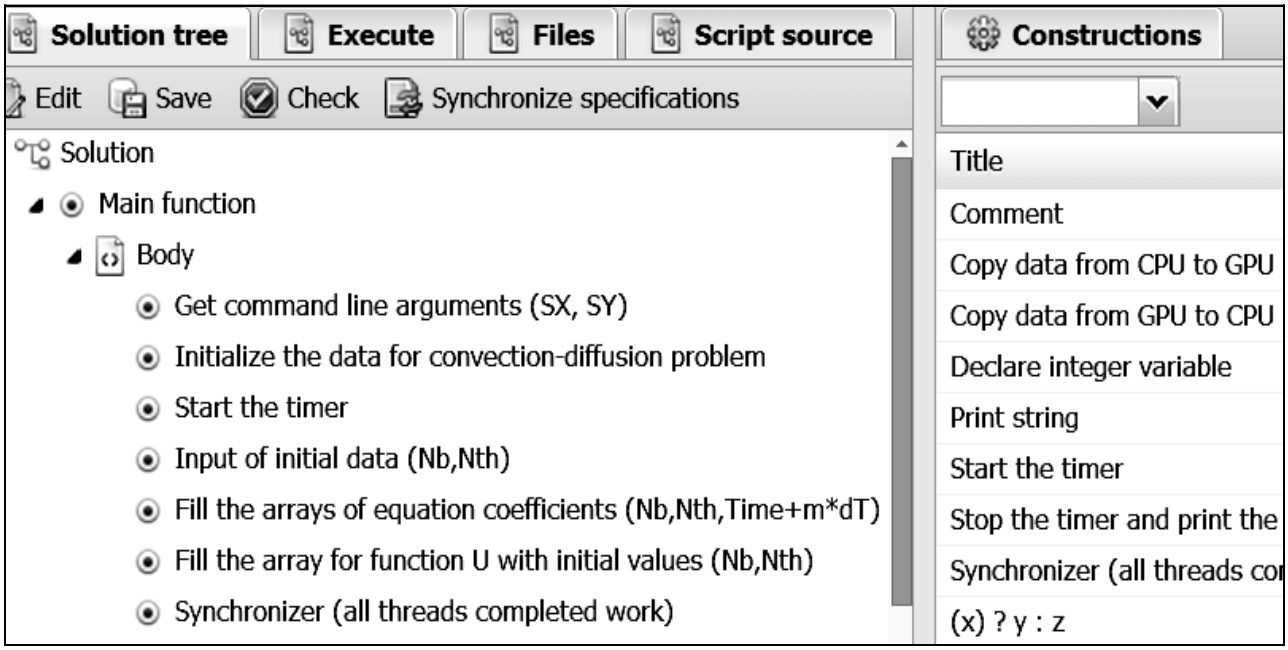

Рис. 2. Фрагмент копії екрану системи ОДСП
з деревом алгоритму та списком конструкцій

конструкції зі списку і додає їх у дерево, використовуючи технологію "drag-and-drop".

Диспетчер є ядром інструментарію, яке організовує зв'язок між клієнтом, генератором програм, шлюзом і базою даних.

Генератор програм виконує генерацію тексту програми на основі схеми алгоритму, побудованої за допомогою клієнтського інтерфейсу.

Шлюз – сервіс, що забезпечує запуск, аналіз та отримання результатів виконання програм у середовищі виконання.

Середовище виконання є програмною платформою, встановленою на стороні сервера системи ОДСП. Середовище включає операційну систему та програмне забезпечення, необхідне для компіляції та запуску програм.

У *базі даних* зберігається інформація про:

- мовні конструкції САА, що використовуються для проектування алгоритмів;
- типи мовних конструкцій;
- розроблені проекти алгоритмів (solutions);
- використовувані цільові мови програмування;
- типи каркасів проектів (frames);
- задачі, виконувані у фоновому режимі;
- типи налаштування запуску проектів.

У базі даних опис кожної мовної конструкції САА включає її подання в природно-лінгвістичній формі; інформацію про тип конструкції; реалізацію (шаблон) обраною мовою програмування; назви змінних та список формальних параметрів.

Каркас проекту (frame) – фрагмент тексту мовою програмування, що використовується як основа ("обгортка") для підстановки коду, згенерованого системою на основі схеми алгоритму. Як правило, каркас включає у себе команди підключення необхідних бібліотек.

Система може виконувати запуск зкомпільованих програм відразу, або виконувати їх у фоновому режимі. У другому випадку задача виконується на сервері за допомогою планувальника операційної системи.

У попередніх роботах система в основному використовувалася для розробки паралельних програм сортування для виконання на CPU. В даній роботі базу даних ОДСП доповнено операціями, орієнтова-

ними на GPU-обчислення, які розглянуті у розділі 1. Приклад застосування системи для розробки паралельної програми для виконання на GPU розглядається у наступному розділі.

3. Приклад використання системи ОДСП

У даному розділі проілюстроване використання САА та системи ОДСП на прикладі розробки CUDA програми для розв'язання двовимірної задачі конвективної дифузії, що виникає при математичному моделюванні циркуляції атмосфери в метеорології.

Згадана тестова задача детально розглянута в роботах [12, 13], і полягає у розв'язанні сукупності рівнянь конвективної дифузії та знаходженні значень залежної функції $u = u(t, x, y)$, де $t \in [0;10]$ – час; $(x, y) \in \Omega$ – координати; $\Omega = [0,1] \times [0,1]$ – двовимірна просторова область визначення задачі. В якості функції u може виступати, наприклад, швидкість вітру.

Розв'язання задачі ґрунтується на застосуванні скінченно-різницевого чисельного методу, наведеного в [13]. У процесі розв'язання область Ω розбивається на S_x та S_y рівномірних кроків (вузлів) по осям x та y відповідно. Таким чином, загальна кількість вузлів становить $S_x \times S_y$.

Розпаралелювання обчислень для даної задачі полягає у розбитті області розмірністю $S_x \times S_y$ на підмножини та паралельному розв'язанні сукупності підзадач, що визначені на цих підмножинах [12].

На рис. 3 показано загальну САА-схему розробленого паралельного GPU-алгоритму для задачі конвективної дифузії. Схема побудована в системі ОДСП у відповідності з методом діалогового конструювання, що описаний у розділі 2. Алгоритм ґрунтується на використанні конструкцій САА, розглянутих у розділі 1.

Алгоритм містить такі основні змінні: `SX` та `SY` – кількість кроків розбиття просторової області по осям `x` та `y` відповідно; `Nb` – кількість блоків у сітці CUDA; `Nth` – кількість потоків у кожному блоці; `Time = 0` – початкове значення часу; `T = 10` – кінцеве значення часу; `dT = 0.001` – часовий крок; `t` – час; `m` – кількість обчислювальних кроків; `h_pX`, `d_pX`, `h_pY`, `d_pY` – масиви аргументів; `Ux` та `Uy` – масиви для зберігання значень функції u.

Схема алгоритму подана у вигляді дерева. Коренева вершина схеми містить текст "`Main function`" і відповідає заголовку функції `main` у програмі мовою С. Тіло основної функції складається із сукупності операторів, які подано як дочірні вершини вузла "`Body`". Реалізація основної функції починається з оператора, який зчитує значення двох параметрів командного рядка, і присвоює відповідні значення змінним `SX` та `SY`. Далі виконується ініціалізація даних і починається відлік часу. Після цього оператори з аргументами `Nb` та `Nth` здійснюють виклики функцій-ядер (наприклад, "`Input of initial data (Nb, Nth)`"). Функції-ядра виконують введення початкових даних, заповнення масивів коефіцієнтів рівнянь та масиву для функції u. Синхронізатор, що наведений після викликів згаданих функцій, виконує затримку обчислень поки всі потоки не завершать роботу. Відмітимо, що алгоритм містить цикл `While`, в тілі якого знаходяться виклики додаткових функцій-ядер, які здійснюють заповнення масивів коефіцієнтів, граничних умов, обчислення значень `Ux`, `Uy` та середнього значення. Алгоритми функцій-ядер подаються в окремих схемах.

Загальним результатом виконання алгоритму є сукупність обчислених значень функції u для координат x та y, де $(x, y) \in \Omega$. В кінці програми виконується порівняння отриманих значень функції u з реальними та обчислюється похибка. Далі значення координат та функції u записуються у файли.

Із використанням ОДСП, на основі розглянутих схем автоматично згенеровано текст програми мовою CUDA C.

На рис. 4 як приклад показано короткий фрагмент згенерованої CUDA програми, який відповідає кільком операторам у середині узагальненої схеми алгоритму, показаної на рис. 3.

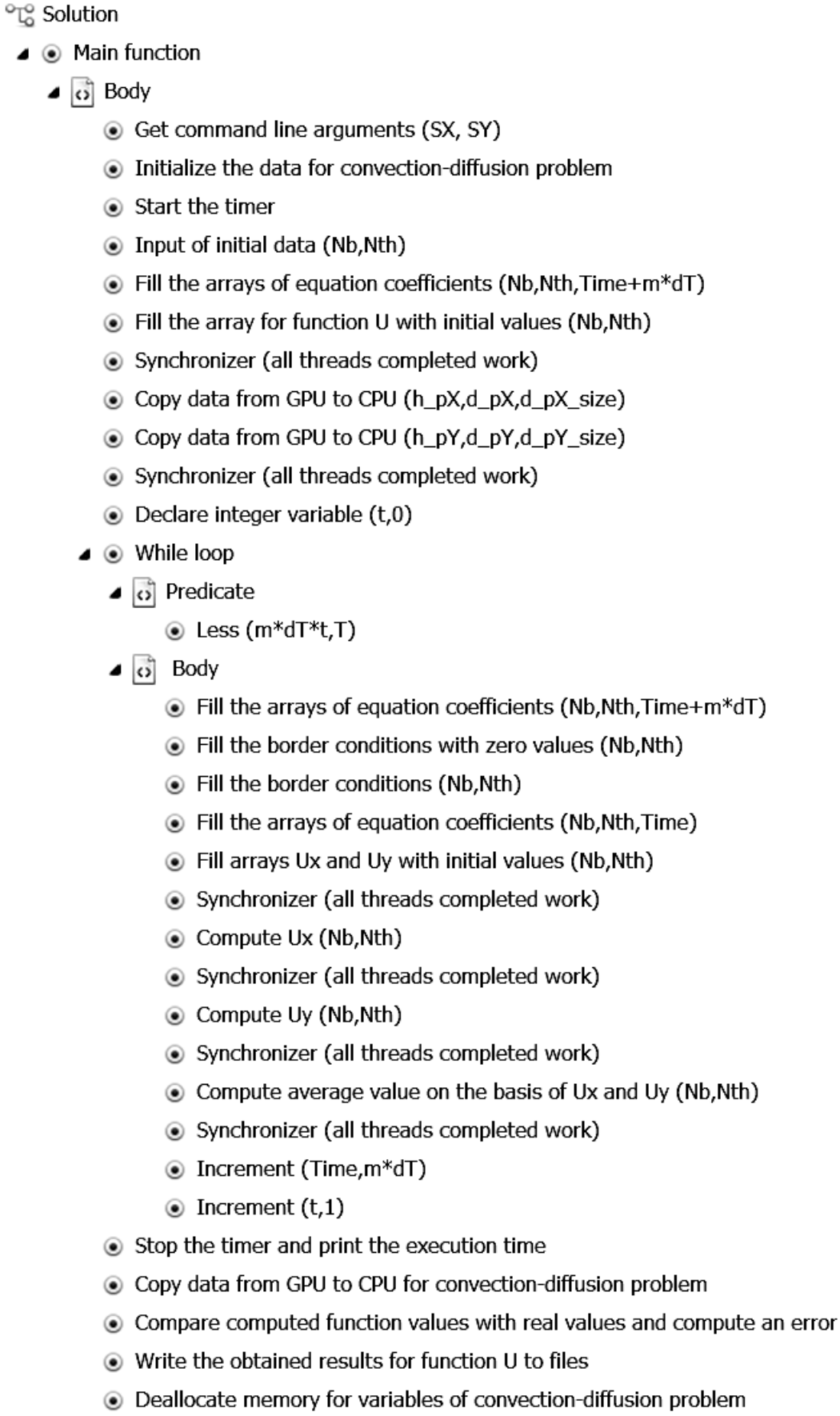

Рис. 3. Загальна схема GPU-алгоритму для задачі конвективної дифузії, побудована в системі ОДСП

```
.....
// Input of initial data (Nb, Nth)
d_Init <<<Nb, Nth>>> (d_pX, d_pY, hx, hy, SX, SY);

// Fill the arrays of equation coefficients
// (Nb, Nth, Time + m * dT)
d_Definition <<<Nb, Nth>>> (d_pX, d_pY, Time+m*dT, d_pV1,
                      d_pW1, d_pV2, d_pW2, d_pF, SX, SY, SIZE);

// Fill the array for function U with initial values
// (Nb, Nth)
d_RealU <<<Nb, Nth>>> (d_pX, d_pY, Time, d_pV1, d_pW1,
                                    d_pV2, d_pW2, d_pU, SIZE);

// Synchronizer (all threads completed work)
CUDAEvent();

// Copy data from GPU to CPU (h_pX, d_pX, d_pX_size)
cudaMemcpy(h_pX, d_pX, d_pX_size, cudaMemcpyDeviceToHost);
.....
```

Рис. 4. Фрагмент згенерованої CUDA програми для задачі конвективної дифузії

Наведений фрагмент програми починається з виклику функції-ядра `d_Init`, що виконує введення початкових даних, і закінчується викликом функції, яка копіює значення змінної `d_Px` у змінну `h_Px` з пам'яті GPU в пам'ять CPU. Перед викликом кожної функції вказано коментарій з текстом відповідної операції САА. У порівнянні зі схемою алгоритму, програмний код містить додаткові змінні, що не були включені в алгоритм з метою його спрощення. Відображення кожної операції САА в текст мовою програмування зазначається в базі даних ОДСП у вигляді параметризованих шаблонів.

Проведено експеримент із виконання розробленої CUDA програми та відповідної послідовної програми на одному з вузлів розділу СКІТ-4 кластеру Інституту кібернетики НАНУ [14]. Згаданий вузол містить графічний прискорювач NVIDIA Tesla M2075 (448 обчислювальних ядер) та центральний процесор Intel Xeon E5-2670. Чисельні експерименти виконано для значень розміру задачі від $S_x \times S_y = 64 \times 64$ до $S_x \times S_y = 2048 \times 2048$. Отримані значення мультипроцесорного прискорення T_s / T_p, де T_s та T_p – час виконання послідовної та паралельної програми відповідно. На рис. 5 показано діаграму залежності мультипроцесорного прискорення від розміру задачі $S_x \times S_y$. Максимальне значення прискорення при $S_x \times S_y = 2048 \times 2048$ становило 69 разів.

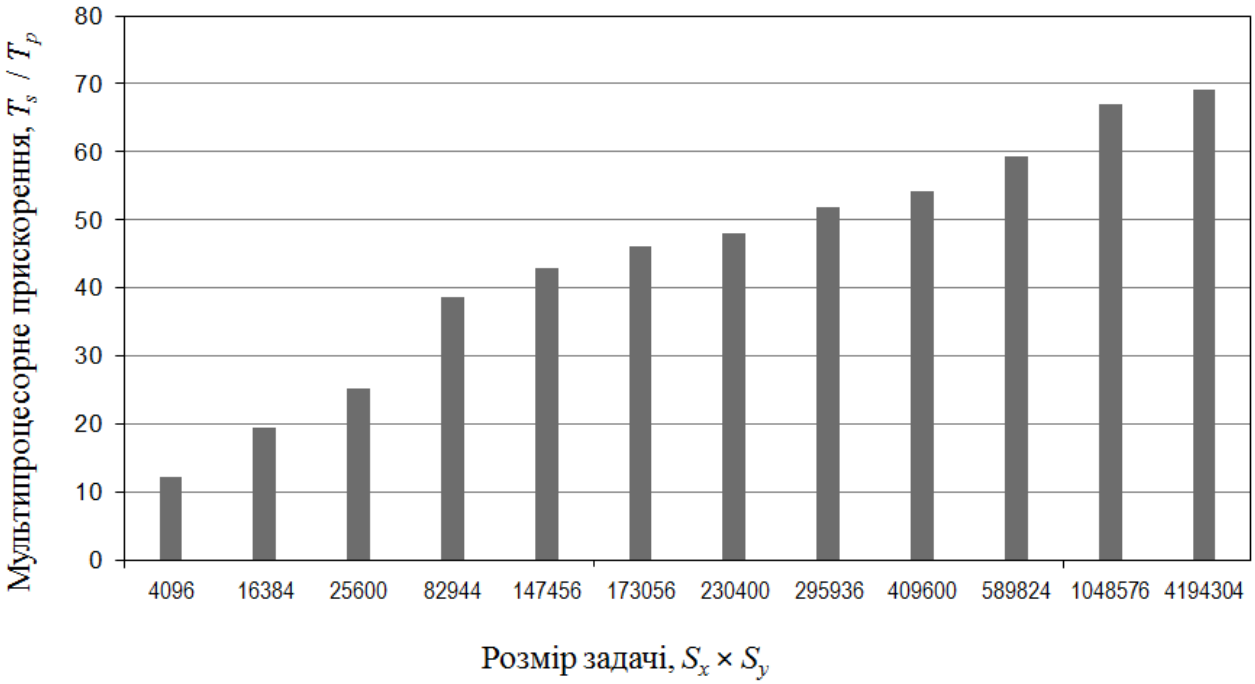

Рис. 5. Залежність мультипроцесорного прискорення T_s / T_p від розміру задачі $S_x \times S_y$ для паралельної програми конвективної дифузії

Висновки

Розроблено алгеброалгоритмічний підхід до автоматизованого проектування та генерації програм для графічних прискорювачів. Перевагою підходу є використання мовних конструкцій, близьких до природної мови, а також застосування методу, який забезпечує синтаксичну правильність алгоритмів та програм, що проектуються. Підхід реалізовано в онлайновому інструментарії, в основу якого покладене діалогове порівневе конструювання алгоритмів та генерація програм на основі компонент повторного використання, у якості яких виступають конструкції алгебри алгоритмів.

Застосування інструментарію проілюстровано на прикладі розробки паралельної програми для задачі конвективної дифузії. Проведено експеримент із виконання розробленої програми на графічному прискорювачі, який показав гарну ступінь розпаралелюваності обчислень.

Перспективи виконаної роботи полягають у подальшому розвитку алгебро-алгоритмічних засобів для опису додаткових можливостей платформи CUDA. Також планується налаштування інструментарію на розробку програм, що використовують інші технології обчислень на GPU, наприклад, OpenCL.

1. *Андон Ф.И., Дорошенко А.Е., Цейтлин Г.Е., Яценко Е.А.* Алгеброалгоритмические модели и методы параллельного программирования. – Киев: Академпериодика, 2007. – 631 с.
2. *Дорошенко А.Е., Жереб К.А., Яценко Е.А.* Формализованное проектирование эффективных многопоточных программ // Проблеми програмування. – 2007. – № 1. – С. 17–30.
3. *Яценко Е.А.* Интеграция инструментальных средств алгебры алгоритмов и переписывания термов для разработки эффективных параллельных программ // Проблеми програмування. – 2013. – № 2. – С. 62–70.
4. *Иовчев В.А., Мохница А.С.* Инструментальные средства алгебры алгоритмики на платформе Web 2.0 // Проблеми програмування. – 2010. – № 2–3. – С. 547–555.
5. *Дорошенко А.Е., Цейтлин Г.Е., Иовчев В.А.* Высокоуровневые средства автоматизации проектирования параллельных алгорит-

мов // Проблеми програмування. – 2009. – № 3. – С. 19–29.
6. *Wilt N.* The CUDA handbook. A Comprehensive Guide to GPU Programming. – Boston: Addison-Wesley, 2013. – 528 p.
7. *Ueng S. et al.* CUDA-lite: Reducing GPU Programming Complexity // Proc. 21st Int. Workshop on Languages and Compilers for Parallel Computing (LCPC'2008). – 2008. – P. 1–15.
8. *Haid W. et al.* Efficient Execution of Kahn Process Networks on Multi-Processor Systems Using Protothreads and Windowed FIFOs // Proc. Workshop on Embedded Systems for Real-Time Multimedia (ESTImedia'09). – 2009. – P. 35–44.
9. *Han T.D., Abdelrahman T.S.* hiCUDA: A High-level Language for GPU programming // IEEE Transactions on Parallel and Distributed systems. – 2011. – Vol. 22, N 1. – P. 78–90.
10. *Jung H., Yi Y., Ha S.* Automatic CUDA Code Synthesis Framework for Multicore CPU and GPU architectures // Lecture Notes in Computer Science. – 2012. – Vol. 7203. – P. 579–588.
11. *Dubach C., Cheng P., Rabbah R., Bacon D.F., Fink S.J.* Compiling a High-Level Language for GPUs (via Language Support for Architectures and Compilers) // Proc. 33rd ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI'12). – 2012. – P. 1–12.
12. *Прусов В.А., Дорошенко А.Ю., Кацалова Л.М., Бекетов О.Г.* Паралельні обчислення двовимірної задачі конвективної дифузії на відеокарті // Комп'ютерне моделювання в хімії, технологіях і системах сталого розвитку – КМХТ-2014: збірник наукових статей IV Міжнар. наук.-практ. конф. – К.: НТУУ "КПІ", 2014. – С. 42–47.
13. *Прусов В.А., Дорошенко А.Е., Черныш Р.И.* Метод численного решения многомерной задачи конвективной диффузии // Кибернетика и системный анализ. – 2009. – № 1. – С. 100–107.
14. *Суперкомп'ютер* Інституту кібернетики НАН України [Електронний ресурс]. – Режим доступу: http://icybcluster.org.ua. – 01.10.2014 р.

Одержано 07.10.2014

***Про авторів*:**

Дорошенко Анатолій Юхимович,
доктор фізико-математичних наук,
професор, завідувач відділу теорії
комп'ютерних обчислень Інституту
програмних систем НАН України,

Бекетов Олексій Генадійович,
аспірант Інституту програмних систем
НАН України,

Іванів Руслан Богданович,
магістрант Національного технічного
університету України "КПІ",

Іовчев Володимир Олександрович,
молодший науковий співробітник
Інституту програмних систем
НАН України,

Мироненко Ігор Олексійович,
магістрант Національного технічного
університету України "КПІ",

Яценко Олена Анатоліївна,
кандидат фізико-математичних наук,
старший науковий співробітник
Інституту програмних систем
НАН України.

Місце роботи авторів:

Інститут програмних систем
НАН України,
03187, Київ-187,
проспект Академіка Глушкова, 40.
Тел.: (044) 526 35 59.
E-mail: dor@isofts.kiev.ua,
beketov.oleksii@gmail.com,
iovchev.v@gmail.com,
oayat@ukr.net.

Національний технічний університет
України "КПІ",
03056, Київ-056,
проспект Перемоги, 37.
E-mail: ivanivruslan@gmail.com,
mioe@ukr.net.