АРХІТЕКТУРА ТА КОМПОНЕНТИ КОМП'ЮТЕРНИХ СИСТЕМ

УДК 681.325

А. Мельник, Н. Козак
Національний університет "Львівська політехніка",
кафедра електронних обчислювальних машин

ВРАХУВАННЯ ОСОБЛИВОСТЕЙ ГРАФІЧНОГО ПРОЦЕСОРА В ПРОЦЕСІ СТВОРЕННЯ ЗАСОБІВ АВТОМАТИЧНОГО РОЗПАРАЛЕЛЕННЯ ПРОГРАМ

Встановлено проблемні аспекти виконання паралельних алгоритмів на графічному прискорювачі. На основі встановлених особливостей запропоновано алгоритм генерації програм для графічного процесора.

Ключові слова: графічний процесор, паралельні обчислення, ОКМД.

This paper describes the main problems in implementing parallel algorithms on graphics accelerators. Based on the established features proposed algorithm generating programs for the GPU.

Key words: GPU, parallel computing, SIMD.

Вступ

GPGPU (*англ. General-purpose graphics processing units*) – технологія використання графічного процесора для виконання довільних обчислень, що зазвичай виконує центральний процесор. Цю технологію зокрема можна використати для побудови персонального суперкомп'ютера [1]. З іншого боку, якщо зважати ще і на той факт, що графічний прискорювач є фактично стандартною складовою персонального комп'ютера, то ця технологія, що не вимагає затрат на купівлю додаткового обладнання, може використовуватися і у звичайному персональному комп'ютері. В такому разі задачу реалізації обчислень GPGPU можна розглядати також як і оптимізацію використання наявних ресурсів ПК.

Аналіз публікацій і окреслення проблеми

Поширення технології GPGPU гальмує складність програмування графічних прискорювачів [2]. Більшість інструментів програмування графічних прискорювачів вимагають переписання програми, що була спочатку написана для центрального процесора. Це зокрема AMD IL, OpenCL, CUDA [3, 4]. Значно спрощують задачу компілятори [5, 6], які підтримують стандарт OpenACC [7]. У такому разі програму, що спочатку була написана для центрального процесора, можна не переписувати. Тим не менше стандарт вимагає використання спеціальних директив компілятору для планування паралельності. Створення систем автоматичного планування паралельності для графічних прискорювачів дозволило б звести до мінімуму зусилля програміста під час адаптації програм, попередньо написаних для центрального процесора.

У процесі пошуку методів автоматичного планування паралельності варто звернути увагу на досвід розроблення систем автоматизованого проектування спеціалізованих процесорів. Було встановлено можливість побудови спеціалізованих процесорів на основі потокового графу алгоритму [8], що був одержаний з опису алгоритму мовою високого рівня. Зокрема показано

велику ефективність системи ХАМЕЛЕОН [9], що ґрунтується на такому принципі. Зручно використати цю систему і під час планування паралельних обчислень для графічного прискорювача, оскільки для виконання обчислень, поданих у формі потокового графу алгоритму, добре підходить паралельна архітектура графічного процесора. Результатом планування паралельності програми для графічного прискорювача буде відображення цього потокового графу алгоритму, який генеруватиме система ХАМЕЛЕОН, наприклад, для 8-точкового ШПФ (рис. 1, а).

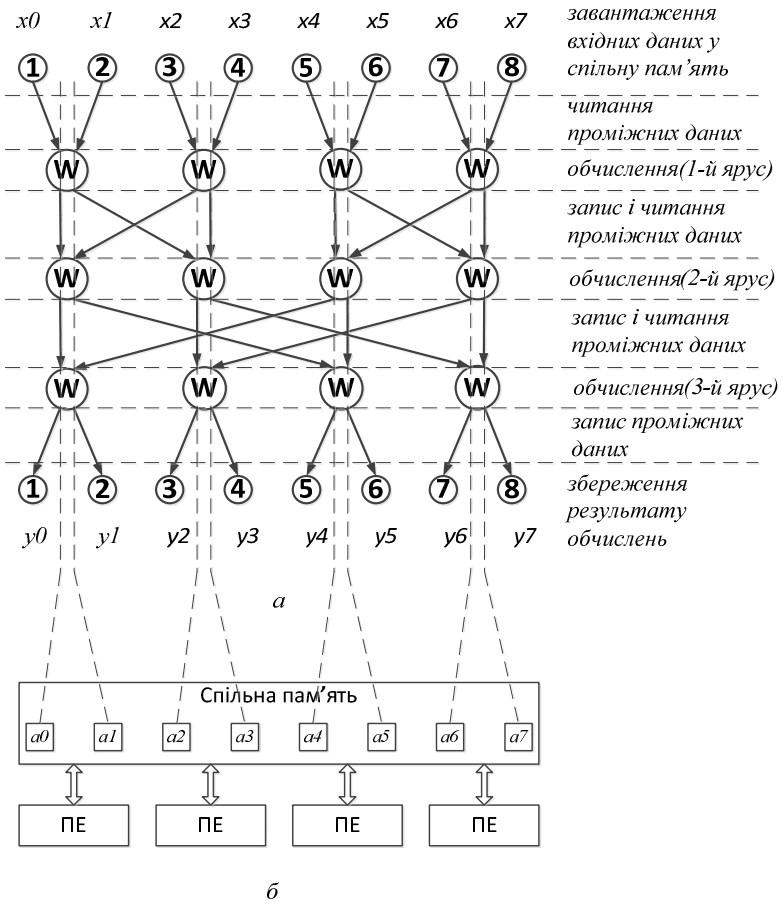

Рис. 1. Відображення потокового графу алгоритму ШПФ для n=8 на структурі із спільною пам'яттю: а – потоковий граф алгоритму ШПФ; б – система типу МКМД

На відміну від реалізації для ПЛІС, де є можливість конфігурації довільної МКМД системи (рис. 1, б), у реалізації для графічного процесора слід зважати на його архітектурні особливості та їх вплив на структуру паралельних обчислень.

Постановка задачі

Метою роботи є розроблення методології адаптації потокового графу алгоритму для ефективного виконання на графічному прискорювачі, яка буде враховувати встановлені архітектурні особливості графічного процесора.

Вибір апаратної платформи для досліджень

Для дослідження методів організації паралельних обчислень на графічному прискорювачі краще використати графічний процесор з універсальнішою архітектурою, тому було обрано графічний адаптер AMD серії Radeon на базі кристала Cayman [3]. На відміну від графічних процесорів виробника NVIDIA [4], у цього графічного процесора наявна глобальна спільна пам'ять (рис. 2, а), також формат інструкцій процесорних елементів графічного процесора Cayman є VLIW4 (рис. 2, в). Оскільки цей графічний процесор володіє декількома моделями паралельності, то і архітектуру його можна вважати універсальнішою відносно різних типів графічних процесорів. Крім того, поєднання моделей паралельності збільшує його гнучкість відносно виконання довільних паралельних обчислень.

Три рівні архітектури графічного процесора Cayman

Архітектуру графічного прискорювача в контексті планування паралельних обчислень можна поділити на три рівні: архітектуру процесорних елементів, архітектуру блоків ОКМД і архітектуру верхнього рівня. На кожному із цих рівнів модель паралельних обчислень має свою специфіку.

Верхній рівень архітектури графічного прискорювача Cayman являє собою 24 незалежні блоки процесорних ядер, які між собою реалізовують повну архітектуру МКМД, але з обмеженою спільною пам'яттю для організації взаємодії між блоками. Враховуючи відносно повільний доступ до цієї пам'яті, цей рівень архітектури не може ефективно виконувати довільні паралельні обчислення з інтенсивним обміном проміжними результатами обчислень. З іншого боку, при низькій інтенсивності обміну даними, функціональність цієї пам'яті цілком достатня для виконання паралельних обчислень (рис. 2, а).

На проміжному рівні архітектура графічного прискорювача являє собою систему, близьку за типом до ОКМД, що об'єднує декілька обчислювальних ядер в один блок. Цей рівень є основним, оскільки графічний прискорювач призначений для виконання насамперед векторних операцій. Блок ОКМД кристала Cayman складається з 16 обчислювальних ядер, які являють собою базові процесорні елементи. Кожний процесорний елемент має доступ до відносно швидкої локальної спільної пам'яті (рис. 2, б). Процесорні елементи в межах блоку можуть одночасно виконувати операції над різними даними, але за допомогою однієї підпрограми.

Самі процесорні елементи виконують команди формату VLIW4 (рис. 2, в), тобто одне командне слово виконує чотири повністю незалежні операції за допомогою окремих АЛП. Хоча цей рівень архітектури графічного процесора і забезпечує найбільшу швидкодію обміну проміжними результатами обчислень, однак ступінь паралельності обмежується чотирма АЛП і лише трьома портами для читання регістрів загального призначення.

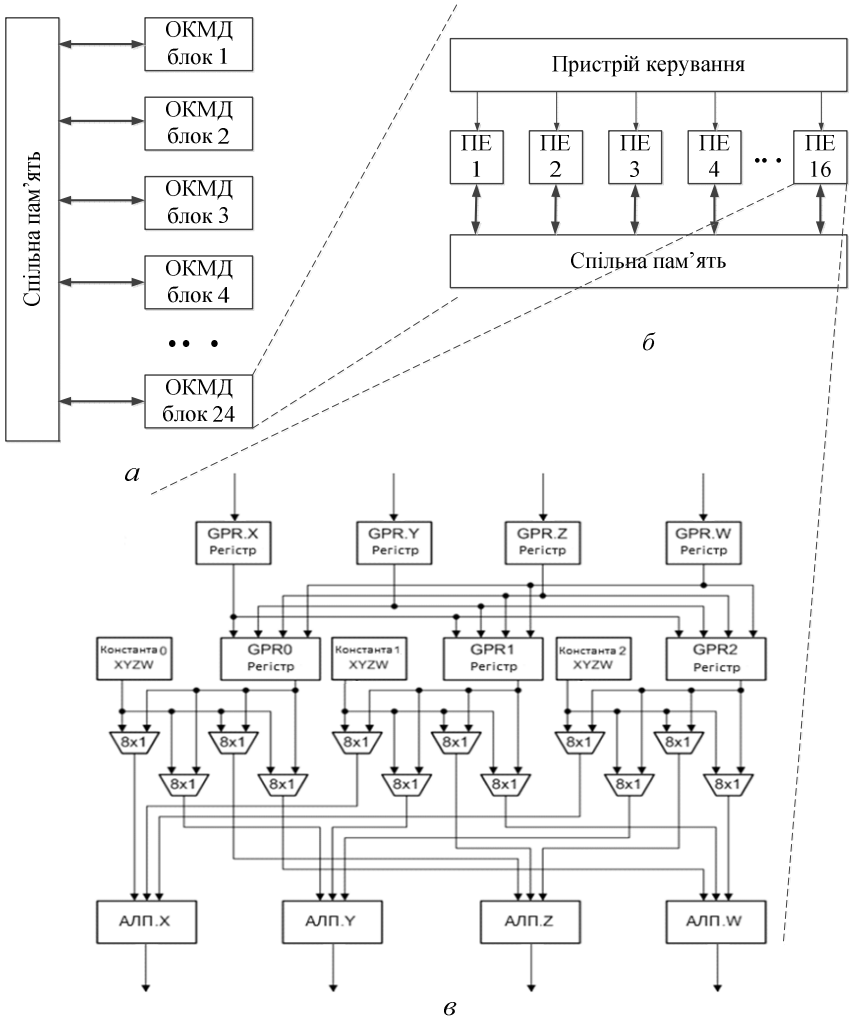

Рис. 2. Три рівні архітектури графічного процесора Cayman: а – архітектура верхнього рівня; б – архітектура блоків ОКМД; в – архітектура процесорних елементів

Виконання програм на графічному прискорювачі

Програми, що виконуються на графічному процесорі, запускаються за допомогою програми, що виконується на центральному процесорі, шляхом взаємодії через API (*англ. application programming interface*). Відповідно для запуску програм, написаних за допомогою AMD IL, використовується функція *calCtxRunProgram()*, а для високорівневого OpenCL – *clEnqueueNDRangeKernel()* [3]. Ці виклики задають лише кількість копій програми для виконання на масиві процесорних елементів, не передаючи інформації про структуру паралельності обчислень. Шляхом читання ідентифікаторів *get_global_id*() і *get_local_id*(), підставляючи значення у відповідний вираз, можна працювати з передбачуваними ділянками спільної пам'яті, що є достатнім для здійснення регулярних векторних обчислень. У випадку реалізації багатоярусного потокового графу алгоритму на момент виконання немає жодної інформації про актуальний ярус обчислень, тому для ідентифікації ярусів доцільно використати маркер ярусу, що збільшується на 1 заданою логікою. Оскільки на усіх процесорних елементах в межах блоку ОКМД виконується одна послідовність команд, то підпрограми вкладаються в одну спільну для всіх процесорних елементів програму. Використовуючи ідентифікатори та маркер ярусу, програма зможе виконувати підпрограму відповідного ярусу. Процесорні елементи, які не виконують цю підпрограму, блокуються і переходять в неактивний стан. Алгоритм вибірки підпрограм наведено на рис. 3.

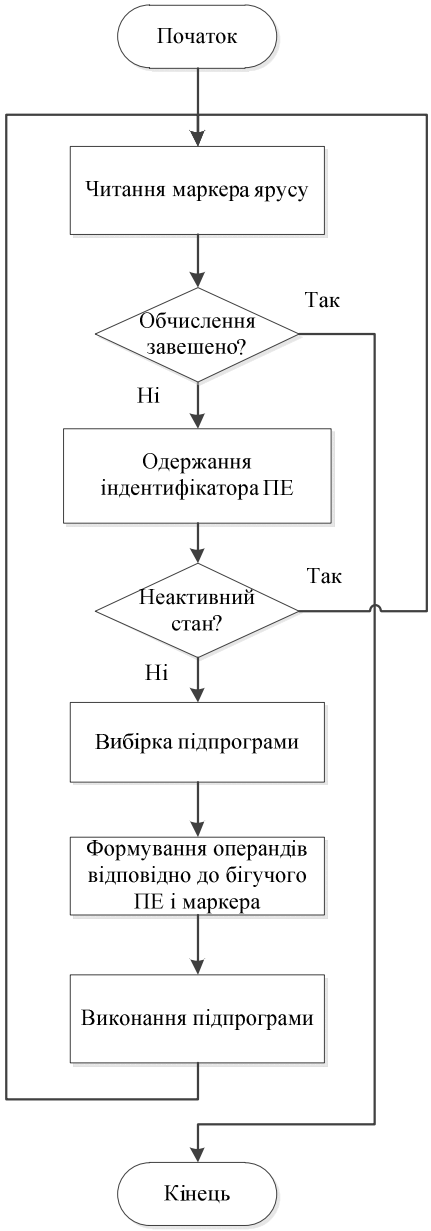

Рис. 3. Алгоритм вибірки підпрограм у програмі для графічного прискорювача

Алгоритм генерації програми для графічного прискорювача

Вищенаведені архітектурні особливості графічного процесора повинні послідовно враховуватися (рис. 4) для одержання ефективного програмного ядра, придатного для виконання на графічному прискорювачі. Оскільки найбільшу гнучкість забезпечує VLIW архітектура процесорних елементів, то першим етапом повинна бути декомпозиція алгоритму на підпрограми – така, щоб підпрограми, які ми отримаємо, добре розпаралелювалися для архітектури VLIW4. На основі отриманих підпрограм, що умовно вважаються базовими операціями, за допомогою системи ХАМЕЛЕОН можна одержати потоковий граф алгоритму. На верхньому рівні обмін даними буде найповільнішим, тому наступним кроком потрібно виконати декомпозицію одержаного потокового графу алгоритму на фракції з шириною ярусу не більшою за 16, які б рідко обмінювалися проміжними результатами між собою, тобто були достатньо незалежними. Далі ці фракції потокового графу алгоритму слід трансформувати відповідно до архітектури ОКМД блоку процесорних елементів так, щоб на кожному з ярусів були лише однакові підпрограми [2].

Рис. 4. Алгоритм генерації програми для графічного прискорювача

Висновки

1. Проаналізовано три рівні архітектури графічного процесора Cayman в контексті планування паралельних обчислень у графічному прискорювачі. Для кожного з трьох рівнів показано обмеження виконання паралельних обчислень.

2. Запропоновано структуру програми для графічного прискорювача, що складається з підпрограм розпаралеленого алгоритму.

3. Відповідно до показаної структури програми і обмежень паралельності графічного процесора запропоновано алгоритм генерації програми для графічного прискорювача на базі потокового графу алгоритму.

1. Мельник В. Стан та перспективи розвитку високопродуктивних обчислювальних систем [Текст] / В. Мельник // Вісник Нац. ун-ту "Львівська політехніка" "Комп'ютерні системи та мережі". – Львів, 2011. – С. 96–104. 2. Козак Н. Реалізація паралельних обчислень в графічних прискорювачах [Текст] / Н. Козак // Conference ACSN-2011. – Львів, 2011, – С. 47-49. 3. AMD HD 6900 Series Instruction Set Architecture [Електронний ресурс] / AMD Режим доступу:http: // developer. amd.com/sdks/amdappsdk/assets/AMD_HD_6900_Series_Instruction_Set_Architecture.pdf. 4. NVIDIA Fermi Compute Architecture Whitepaper [Електронний ресурс] / NVIDIA Режим доступу http://www. nvidia.com/content/PDF/ fermi_white_papers/NVIDIA Fermi Compute_Architecture_Whitepaper.pdf. 5. PGI Accelerator Programming Model for Fortran & C [Електронний ресурс] / PGI Режим доступу : http://www.pgroup.com/lit/whitepapers/pgi_accel_prog_model_1.3.pdf. 6. CAPS OpenACC Compiler [Електронний ресурс] / CAPS Режим доступу : http://www.caps-entreprise.com/wp-content/uploads/ 2012/07/CAPS_PROD_EN_openacc_201208.pdf. 7. The OpenACC Application Programming Interface [Електронний ресурс] / CRAY, CAPS, PGI, NVIDIA Режим доступу : http://www.openacc.org/ sites/default/files/OpenACC.1.0_0.pdf. 8. Melnyk, A., Automatic generation of ASICs [Текст] / A. Melnyk, A. Salo // NASA-ISA Conference AHS-2007, Edinburgh, 2007. – С. 96-101. 9. Мельник А. ХАМЕЛЕОН – система високорівневого синтезу спеціалізованих процесорів [Текст] / Мельник А.О., Сало А.М., Клименко В. [та ін.] // Наук.-техн. журн. Національного аерокосмічного університету ім. М.Є. Жуковського "Харківський авіаційний інститут". – 2009. – № 5. – С. 189–195.