УДК 681.3, УДК 004.02

А. О. Мельник, Н. Б. Козак
Національний університет “Львівська політехніка”,
кафедра електронних обчислювальних машин

ПІДХОДИ ДО ПРОГРАМУВАННЯ ПРИСКОРЮВАЧІВ ОБЧИСЛЕНЬ

Розглянуто проблему програмування прискорювачів обчислень, виділено шість підходів до їх програмування. Визначено підхід анотування фрагментів коду програми без інформації про трансформацію як найперспективніший для подальшого розвитку. Показано напрям розвитку цього підходу на основі автоматизації розпаралелення фрагментів коду.

Ключові слова: прискорювачі обчислень, технології програмування прискорювачів, паралелізм.

APPROACHES TO COMPUTING ACCELERATORS PROGRAMMING

The problem of computing accelerators programming is considered. Six approaches to the accelerators programming are identified. An approach to the program snippets annotation without information about transformation as most promising for further development is defined. The direction of this approach development based code snippets automatic parallelization is shown.

Key words: computing accelerators, techniques of accelerators programming, parallelism.

Вступ

Для прискорювачів обчислень, що спочатку переважно будувалися на основі спеціалізованих процесорів цифрових сигналів [1, 2], програмне забезпечення розроблялося як невід’ємна складова частина проекту, оскільки їх робота завжди була націлена на розв’язання лише певних задач предметної області. Сучасні ж прискорювачі обчислень компаній AMD, Intel та NVIDIA здатні виконувати довільні обчислення, прискорення яких досягається розвиненим паралелізмом обчислень. Відповідно, гнучкість таких систем на порядок вища і дає змогу застосовувати ширший спектр технологій програмування.

Постановка задачі

Сучасні технології програмування прискорювачів обчислень відрізняються складністю та способом використання [3]. Складність реалізації для них засобів програмування також різна. Для вибору перспективних концепцій подальшого розвитку технологій програмування прискорювачів необхідно виконати порівняльний аналіз технологій з виокремленням спільних підходів, які застосовуються для підмножин цих технологій. Отже, під час такого аналізу потрібно виконати такі завдання:

- виокремити основні підходи до програмування прискорювачів обчислень;
- провести порівняння підходів;
- показати напрям розвитку технологій програмування прискорювачів обчислень.

Підходи до програмування прискорювачів

Виконавши огляд сучасних технологій програмування прискорювачів, можна виокремити кілька основних підходів, умовно розділивши їх на дві групи. До першої групи зарахуємо класичні підходи, серед яких використання:

- спеціалізованих технологій програмування;
- бібліотеки шаблонів;
- прикладних бібліотек предметної області.

Зручніша друга група підходів, що має на меті спростити процес використання вже розробленого програмного забезпечення, що спочатку було написане для центрального процесора. До таких підходів зарахуємо:

- використання спеціалізованого компілятора з автоматичним розпаралеленням;
- анотування фрагментів коду програми з інформації про трансформацію;
- анотування фрагментів коду програми без інформації про трансформацію**.**

Спеціалізовані технології програмування. Спеціалізовані технології програмування прискорювачів, як правило, з'являються практично одночасно із самими прискорювачами. Технологія програмування CUDA [4] фактично є синонімом відповідної апаратної платформи, а OpenCL майже відразу почали використовувати для програмування графічних прискорювачів компанії AMD [5]. Водночас, програмування за допомогою спеціальних технологій програмування, що майже завжди реалізовує свою специфічну модель програмування, потребує глибокого знання архітектури прискорювача обчислень.

Бібліотеки шаблонів. У разі використання бібліотеки шаблонів, що реалізовані для конкретної платформи прискорювача, передбачається абстрагування від архітектури апаратних засобів. Паралелізм стандартних алгоритмів вже закладений в самій бібліотеці. Загалом, всі такі бібліотеки подібні за структурою до STL. До них належать TBB [6], Thrust [7], Bolt [8], C++ AMP [9].

Прикладні бібліотеки. Програмування з використанням прикладних бібліотек, що реалізують розв'язання задач певної предметної області, є порівняно зручним підходом. У випадку реалізації для прискорювачів у прикладних бібліотеках також закладений паралелізм алгоритмів. До прикладних бібліотек для прискорювачів обчислень можна зарахувати clBLAS [10], clFFT [11], CUBLAS [12], CUSPARCE [13], CUFFT [14], CURAND [15].

Спеціалізований компілятор з автоматичним розпаралеленням. Розроблення повністю функціонального спеціалізованого компілятора з автоматичним розпаралеленням з мов загального призначення (C/C++, FORTRAN, C#, Java) для конкретної платформи прискорювача є доволі складним завданням. Загалом, такий компілятор повинен реалізувати такі основні завдання:

- розподіл навантаження між центральним процесором і прискорювачем;
- автоматичне розпаралелення;
- трансляція паралельних обчислень відповідно до програмної моделі прискорювача.

Процес ***розподілу навантаження між центральним процесором і прискорювачем*** полягає у виділенні у вхідному алгоритмі таких фрагментів, які придатні для ефективного розпаралелення і виконання на прискорювачі. Використовуючи закон Амдала, можна оцінити максимальне прискорення за цього розподілу так:

$$S' = \frac{1}{1 - \sum_{k=1}^{K} \left(a_k + \frac{a_k}{n_k} \right)}, \tag{1}$$

де K – кількість фрагментів для розпаралелення; a_k і n_k – частка і ступінь розпаралелення кожного фрагмента.

Важливим завданням на цьому етапі є також вибір механізму обміну проміжними результатами обчислень. Такий обмін сповільнюватиме виконання алгоритму:

$$S'' = \frac{t}{t + \sum_{k=1}^{K} (t_{ik} + t_{ok})}, \tag{2}$$

де t – початковий час виконання алгоритму; t_{ik} і t_{ok} – час вводу і виводу даних для кожного фрагмента.

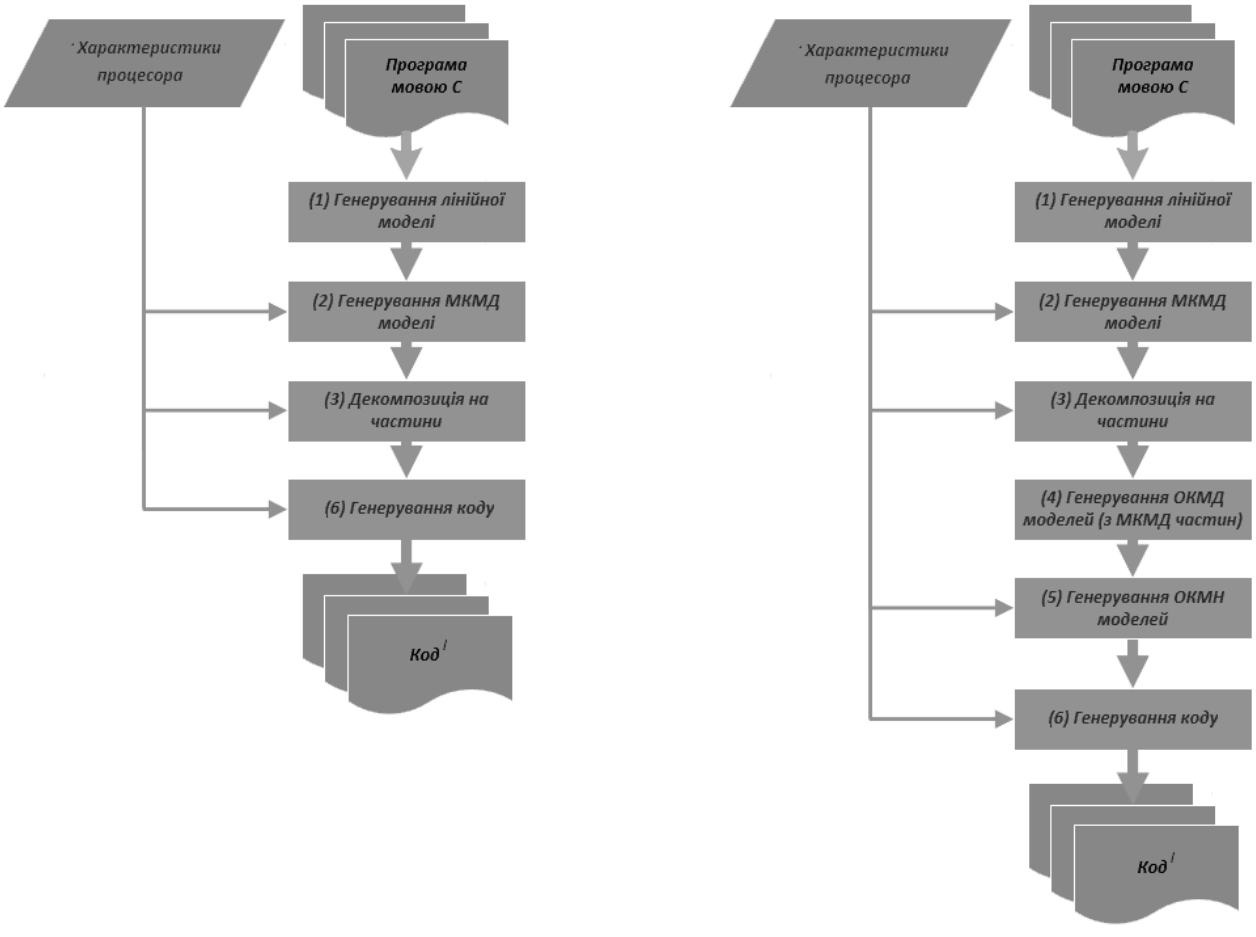
Для ефективного прискорення вимогою буде співвідношення:

$$S' > \frac{S'''}{S''}, \tag{3}$$

де S''' – деяке мінімальне прискорення, за якого використання додаткового обладнання є доцільним

В процесі розподілу навантаження між центральним процесором і прискорювачем найчастіше використовують евристичні алгоритми. Аналіз їх варіативності та ефективності є предметом дослідження [16].

Проблема ***автоматичного розпаралелення*** є ключовою задачею такого спеціалізованого компілятора. Потрібно також зазначити, що у більшості випадків, зокрема і для графічних прискорювачів, потрібно виконувати трансформацію на проміжні ОКМД моделі *(рис. б)*. Іншим важливим завданням є укрупнення базових операцій *(заміни типу зернистості)*, внаслідок чого ефективність виконання паралельних обчислень на прискорювачах суттєво змінюється через особливості роботи підсистеми пам'яті.



а *б*

Автоматичне розпаралелення програмного коду:
а – розпаралелення для МКМД-подібних моделей обчислень;
б – розпаралелення для ОКМД-подібних моделей обчислень

Задачу розпаралелення простих фрагментів коду можна ефективно розв'язати [17], але у випадку доволі складного програмного коду виникають проблеми:

- проблема виклику підпрограм;
- проблема створення локальних копій даних змінного розміру;
- проблема взаємодії з ресурсами системи;
- інші проблеми програмних невизначеностей часу виконання.

Побудову систем для ефективного розпаралелення складних фрагментів коду ще не здійснено, тому сьогодні активно ведуться дослідження в цьому напрямі [18].

Трансляція паралельних обчислень відповідно до програмної моделі прискорювача є задачею, реалізація якої істотно залежить від типу вибраного прискорювача**.** Крім архітектури Intel MIC [19], в основі якої класичний набір команд архітектури x86, типи архітектури більшості прискорювачів доволі своєрідні й мають свій спеціалізований набір команд. Як правило, у побудові компіляторів для прискорювачів використовується складна інфраструктура програмних інструментів на основі LLVM [20] та Open64 [21]. Іноді організація роботи прискорювачів передбачає доволі специфічні механізми. Так, приміром, для графічних прискорювачів передбачений механізм компіляції програмного коду безпосередньо драйвером пристрою [22].

Анотування фрагментів коду програми з інформацією про трансформацію. Оскільки розроблення спеціалізованих компіляторів для прискорювачів з автоматичним розпаралеленням є надскладною задачею, то проміжним етапом на шляху до створення таких компіляторів є метод анотування фрагментів коду програми з інформацією про розпаралелювальну здатність цього фрагмента. У випадку графічних прискорювачів такий підхід реалізують компілятори [23, 24] з підтримкою стандарту OpenACC [25]. По суті, цей стандарт багато в чому повторює принципи технології OpenMP, але спеціалізований для застосування під час програмування прискорювачів. Розв'язання задачі автоматичного розпаралелення відповідно до цього підходу є частковим, оскільки потребує від програміста використання спеціальних директив у коді програми.

Анотування фрагментів коду програми без інформації про трансформацію. У ході досліджень з розроблення автоматичних засобів розпаралелення програм для графічних прискорювачів [18] найкращих результатів досягнуто для автоматичного розпаралелення окремих ділянок коду, тому підхід анотування фрагментів коду для розпаралелення можна покращити, вилучивши з нього інформацію про трансформацію. Суть пропонованого підходу полягає також в анотуванні коду програми, але без інформації про трансформацію. Відповідно до цього підходу програміст також використовує спеціальні директиви для розпаралелення, але без будь-яких атрибутів про паралельність фрагмента коду. Водночас, таке рішення вимагає від програміста ручного розподілу навантаження.

Порівняння підходів до програмування прискорювачів

Оцінка підходів до програмування прискорювачів проводилася за такими основними характеристиками:

- спосіб використання;
- складність використання;
- універсальність;
- складність реалізації засобів програмування.

Порівняння розглянутих підходів до програмування прискорювачів, що наведено в таблиці, виконувалося з урахуванням того, що переважно під час програмування доводиться мати справу з програмним кодом, початково написаним для центрального процесора.

Порівняння підходів до програмування прискорювачів

Підхід	Спосіб використання	Складність використання	Універсальність	Складність реалізації засобів програмування
1	2	3	4	5
Використання спеціалізованих технологій програмування	переписування програмного коду	висока складність	універсальний	порівняно низька
Використання бібліотек шаблонів	заміна частини програмного коду	середня складність	орієнтований на стандартні алгоритми	середня складність
Використання прикладних бібліотек	заміна частини програмного коду	середня складність	орієнтований на прикладну область	середня складність

Продовження таблиці

1	2	3	4	5
Використання спеціалізованих компіляторів з автоматичним розпаралеленням	проста компіляція	відсутня	універсальний	надвисока складність
Анотування фрагментів коду програми з інформацією про трансформацію	директивні анотації компілятору	середня складність	універсальний	висока складність
Анотування фрагментів коду програми без інформації про трансформацію	директивні анотації компілятору	низька складність	універсальний	висока складність

Незважаючи на те, що найзручнішим є підхід використання компілятора з автоматичним розпаралеленням, з погляду простоти реалізації засобів програмування підходи на основі анотування фрагментів коду є компромісними. Ці підходи складніші у використанні, але реалізація засобів програмування для них простіша. Водночас, з-поміж анотування фрагментів коду з інформацією про трансформацію та анотування фрагментів коду без інформації про трансформацію можна виділити другий підхід, який простіший у використанні. Цей підхід має хороші перспективи подальшого розвитку з удосконаленням методів розпаралелення складних програмних конструкцій і може стати проміжною ланкою на шляху до створення спеціалізованих компіляторів з автоматичним розпаралеленням.

Висновки

Аналізуючи сучасні технології програмування прискорювачів, можна зробити такі висновки:

1. Сучасні технології програмування прискорювачів реалізують один з шести підходів: *використання спеціалізованих технологій програмування*, *використання бібліотек шаблонів*, *використання прикладних бібліотек*, *використання спеціалізованих компіляторів з автоматичним розпаралеленням*, *анотування фрагментів коду програми з інформацією про трансформацію* та *анотування фрагментів коду програми без інформації про трансформацію*.

2. Найзручніший підхід, який полягає у використанні спеціалізованого компілятора з автоматичним розпаралеленням, потребує розвинених засобів програмування. Складність реалізації таких засобів програмування є надвисокою.

3. Підхід анотування фрагментів коду без інформації про трансформацію є найперспективнішим. Розвиватиметься підхід за рахунок вдосконалення методів розпаралелення складних програмних конструкцій.

1. Аллен Дж. Архитектура процессоров для цифровой обработки сигналов // ТИИЭР. – 1986. – Т. 73. – № 5. – С. 3–37. 2. Параллельная обработка информации. Параллельные методы и средства распознавания образов. – К.: Наукова думка, 1985. – Т. 2. – 279 с. 3. David Tarditi, Sidd Puri, and Jose Oglesby. Accelerator: simplified programming of graphics-processing units for general-purpose uses via dataparallelism. Technical Report MSR-TR-2004-184, Microsoft Corporation, December 2005. 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. 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. 6. Learn about the Intel Threading Building Blocks library [Електронний ресурс] / Intel. – Режим доступу: https://www.threadingbuildingblocks.org/intel-tbb-tutorial. 7. Thrust [Електронний ресурс] / NVIDIA. – Режим доступу: http://docs.nvidia.com/cuda/thrust/. 8. Bolt C++ Template Library [Електронний ресурс] / AMD. – Режим доступу: http://developer.amd.com/tools-and-sdks/opencl-zone/bolt-c-template-library. 9. C++ Accelerated Massive Parallelism [Електронний ресурс] / Microsoft. – Режим доступу: https://msdn.microsoft.com/ru-ru/library/hh265137. 10. clBLAS Compute Libraries [Електронний ресурс] / AMD. – Режим доступу: http://developer.amd.com/tools-

and-sdks/opencl-zone/acl-amd-compute-libraries/amd-accelerated-parallel-processing-math-libraries/# clBLAS. 11. clFFT Compute Libraries [Електронний ресурс] / AMD. – Режим доступу: http://developer.amd.com/tools-and-sdks/opencl-zone/acl-amd-compute-libraries/amd-accelerated-parallel-processing-math-libraries/#clFFT. 12. cuBLAS [Електронний ресурс] / AMD. – Режим доступу: http://docs.nvidia.com/cuda/cublas. 13. cuSPARCE [Електронний ресурс] / NVIDIA. – Режим доступу: http://docs.nvidia.com/cuda/cusparce. 14. cuFFT [Електронний ресурс] / NVIDIA. – Режим доступу: http://docs.nvidia.com/cuda/cufft. 15. cuRAND [Електронний ресурс] / NVIDIA. – Режим доступу: http://docs.nvidia.com/cuda/curand. 16. Мельник В. А. Самоконфігуровні апаратні прискорювачі обчислень в комп'ютерах / В. А. Мельник, З. Т. Сарайрех // Вісник Національного університету „Львівська політехніка" "Комп'ютерні системи та мережі". – 2010. – № 688. – С. 163–171. 17. Козак Н. Реалізація паралельних обчислень в графічних прискорювачах [Текст] / Н. Козак // Conference ACSN-2011. – Львів, 2011. – С. 47–49. 18. Мельник А. Врахування особливостей графічного процесора в процесі створення засобів автоматичного розпаралелення програм / А. О. Мельник, Н. Б. Козак // Комп'ютерні науки та інформаційні технології: Вісник Національного університету "Львівська політехніка". – 2013. – № 751. – С. 3–8. 19. Архитектура Intel® Many Integrated Core – расширенные возможности [Електронний ресурс] / Intel. – Режим доступу: http://www.intel.ru/content/www/ru/ru/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html. 20. The LLVM Compiler Infrastructure [Електронний ресурс] / LLVM Developer Group. – Режим доступу: http://llvm.org/docs/. 21. Open64 [Електронний ресурс] / Silicon Graphics, Inc., Institute of Computing Technology, Chinese Academy of Sciences, Hewlett Packard, University of Delaware. – Режим доступу: http://sourceforge.net/projects/open64/. 22. Jungwon Kim. SnuCL: an OpenCL Framework for Heterogeneous CPU/GPU Clusters / Jungwon Kim, Sangmin Seo, Jun Lee, Jeongho Nah, Gangwon Jo, Jaejin Lee. // In ICS '12: Proceedings of the 26th International Conference on Supercomputing. – P. 341–352, San Servolo Island, Venice, Italy, Jun. 2012. 23. 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. 24. CAPS OpenACC Compiler [Електронний ресурс] / CAPS. – Режим доступу: http://www.caps-entreprise.com/wp-content/uploads/2012/07/CAPS_PROD_EN_openacc_201208.pdf. 25. 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.