УДК 004.421

Чередніченко А.К., магістрант *(Національний технічний університет України «КПІ»)*

МЕТОДИКА ЕФЕКТИВНОЇ ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ OpenCL ДЛЯ ГЕТЕРОГЕННИХ КОМП'ЮТЕРНИХ СИСТЕМ

Чередніченко А.К. Методика ефективної генерації програмного коду OpenCL для гетерогенних комп'ютерних систем. Сучасні графічні процесори (GPU) є пристроями SIMD архітектури, які надають своїм користувачам значні обчислювальні можливості. OpenCL (Open Computing Language) є основою для написання програм, які виконуються на гетерогенних платформах, та полегшує складні обчислення загального призначення. Слід зазначити, що ручна розробка ефективного коду OpenCL є досить складною задачею. Устатті описана автоматична система генерації OpenCL коду із послідовного коду C. Пропонується загальний підхід для його ефективної генерації. Швидкість виконання результуючої згенерованої OpenCL програми значно вища за швидкість вихідної послідовної програми C.

Ключові слова: ГЕТЕРОГЕННА КОМП'ЮТЕРНА СИСТЕМА, ГРАФІЧНИЙ ПРОЦЕСОР, OpenCL, CPU, GPU, GPGPU, ГЕТЕРОГЕННА ПЛАТФОРМА

Чередниченко А.К. Методика эффективной генерации программного кода OpenCL для гетерогенных компьютерных систем. Современные графические процессоры (GPU) являются устройствами SIMD архитектуры, которые предоставляют своим пользователям значительные вычислительные возможности. OpenCL (Open Computing Language) является основой для написания программ, выполняемых на гетерогенных платформах, и облегчает сложные вычисления общего назначения. Следует отметить, что ручная разработка эффективного кода OpenCL является достаточно сложной задачей. В статье описана автоматическая система генерации OpenCL кода из последовательного кода C. Предлагается общий подход для эффективной генерации такого кода. Скорость выполнения результирующей сгенерированной OpenCL программы значительно выше скорости исходной последовательной программы C.

Ключевые слова: ГЕТЕРОГЕННАЯ КОМПЬЮТЕРНА СИСТЕМА, ГРАФИЧЕСКИЙ ПРОЦЕССОР, OPENCL, CPU, GPU, GPGPU, ГЕТЕРОГЕННАЯ ПЛАТФОРМА.

Cherednichenko A.K. Method of effective OpenCL code generation for heterogeneous computing systems. Graphics Processing Units (GPUs) are SIMD devices which offer tremendous computational power. OpenCL (Open Computing Language) is a framework for writing programs that execute across heterogeneous platforms, facilitating high performance implementations of general-purpose computations. However, manual development of high-performance OpenCL code is rather complicated. This paper describes an automatic code transformation system that generates parallel OpenCL code from input sequential C code. The general approach for effective code generation is proposed. The result performance of the automatically generated OpenCL code in this case is considerably better than the benchmarks' performance on a multicore Central Processing Unit (CPU).

Keywords: HETEROGENEOUS COMPUTER SYSTEM, GRAPHICS PROCESSING UNITS, OPENCL, CPU, GPU, GPGPU, HETEROGENEOUS PLATFORM

Вступ. В останні роки інтенсивний розвиток отримала специфічна галузь високопродуктивних обчислень – використання гетерогенних комп'ютерних систем з паралельними акселераторами, як додатковими пристроями, які приймають на себе значну частину обчислювального навантаження програмного додатку. Гетерогенні комп'ютерні системи з графічними процесорами представляють собою системи на базі SIMD-архітектури. Такі системи складаються з одного командного процесора та декількох модулів обробки даних і характеризуються наявністю одного потоку команд та багатьох потоків даних.

Починаючи з 2003 року, активні дослідження проводяться в області використання сучасних графічних процесорів GPU (Graphics Processing Unit) для вирішення складних обчислювальних задач. Цей напрямок наукових досліджень отримав назву GPGPU (General-Purpose computing on Graphics Processing Units) [1]. Було доведено, що на графічних процесорах доцільно виконувати задачі, для яких раніше традиційно використовувались суперкомп'ютерні архітектури.

Для програмування графічного процесора може бути використаний відкритий кросплатформений стандарт OpenCL (Open Computing Language) [2], який базується на мові програмування **C**. На жаль, такий підхід має ряд обмежень, найбільш ваговим з яких є необхідність вивчення нової мови програмування розробником програмного забезпечення та досить високі вимоги до його професійної кваліфікації, що неодмінно призводить до зростання ціни програмного продукту. Крім того використання спеціалізованої мови

програмування для кожного окремого обчислювального пристрою порушує принципи абстракції та свідчить про невдалий дизайн архітектури системи в цілому.

У зв'язку з очевидною ефективністю виконання на графічних процесорах програм, які добре розпаралелюються, та враховуючи описані обмеження, зростає інтерес до систем автоматичної генерації коду для GPU. У рамках цієї роботи розглянута методика ефективної генерації коду OpenCL для пристроїв SIMD-архітектури. Базуючись на ній, був розроблений відповідний програмний продукт, що підтвердило її ефективність на практиці.

Дана робота базується на дослідженнях, які проводяться у науково-дослідному інституті в місті Ростов-на-Дону. Групою науковців була розроблена відкрита розпаралелюча система (ВРС), яка орієнтована на розробку розпаралелюючих компіляторів та систем автоматичного розпаралелення [3]. При розробці системи для автоматичної генерації коду були використані методики побудови графових моделей програм, впроваджених в ВРС.

Надамо деякі термінологічні пояснення.

Гетерогенна система – це комп'ютерна система, яка складається зрізних обчислювальних модулів. Обчислювальним модулем може бути процесор загального призначення CPU(Central Processing unit), спеціалізований процесор SPP (Special-Purpose Processor): цифровий сигнальний процесор DSP (Digital Signal Processor), графічний процесор GPU (Graphics Processing Unit*)*, спеціалізовані інтегральні схеми ASIC (Application-Specific Integrated Circuit) тощо. В даній роботі під гетерогенною розуміється система з GPU.

GPGPU (General-PurposecomputingonGraphicsProcessingUnits) – використання графічних процесорів для вирішення неграфічних задач.

Постановка завдання. Завданням даної роботи є створення методики ефективної генерації програмного коду OpenCL та його подальшого виконання на графічному процесорі GPU. Основним критерієм ефективності у цьому випадку вважатиметься зростання загальної продуктивності гетерогенної системи. Для цього необхідно правильно виконати розподіл програмного коду між центральним та графічним процесорами. При цьому послідовний код має бути обов'язково виконаний на центральному процесорі, а ділянки коду, що добре розпаралелюються, на графічному процесорі.

Крім того необхідно здійснити дослідження оцінки швидкодії базових алгоритмічних задач та доцільності виконання задач, що розпаралелюються, на графічному процесорі. На базі розробленої методики слід створити програмний продукт та довести або ж спростувати допустимість використання такої методики на практиці.

Виклад основного матеріалу дослідження. Представимо методику ефективної генерації програмного коду OpenCL. Для цього побудуємо відповідну схему алгоритму

(рис. 1) для наступної послідовності етапів генерації коду для GPU:

- Виділення ділянок коду, що в загальному випадку можуть бути розпаралелені.
- Визначення об'єктивної оцінки ефективності обчислень ділянки програмного коду на графічному процесорі.
- Побудова графових моделей програми - граф інформаційних зв'язків, решітковий граф, управляючий граф програми.
- Прийняття рішення щодо можливості розпаралелення даної ділянки коду відповідно до типу пристрою, на якому цей код має бути виконаний (у даному випадку ми розглядаємо виконання коду тільки на центральному або ж графічному процесорах).
- Якщо ділянка коду буде виконуватись на графічному процесорі відбувається виділення пам'яті GPU, копіювання даних з пам'яті CPU у пам'ять GPU, створення обчислювального контексту, генерація функції-ядра GPU, виконання згенерованого ядра, копіювання результуючих даних з пам'яті GPU у пам'ять CPU.

Розглянемо детальніше кожен з виділених етапів.

На першому кроці необхідно визначити ділянки коду, які можуть бути у загальному випадку розпаралелені. До таких можна віднести фрагменти коду, що містять цикли FOR, WHILE, DO-WHILE.

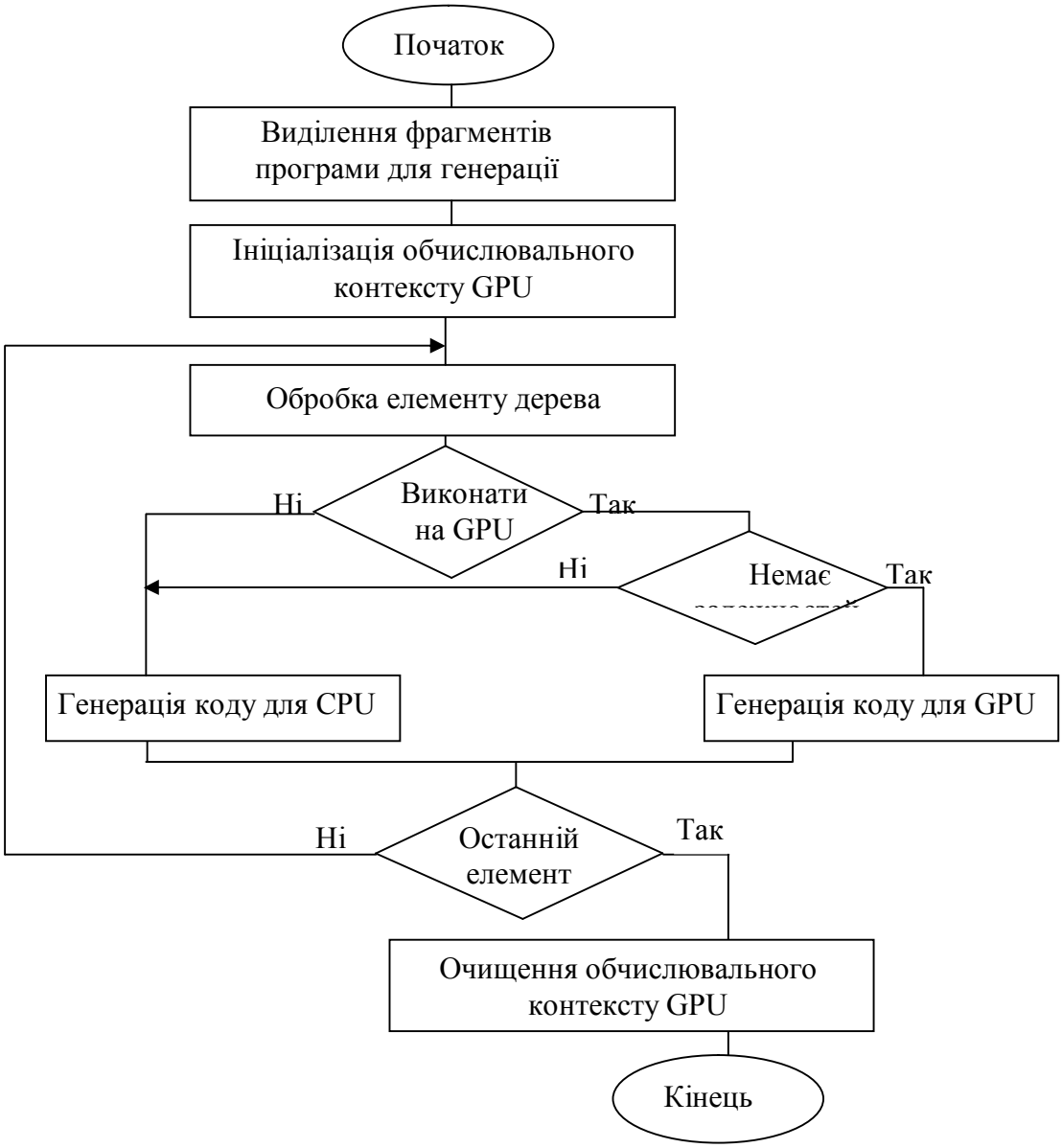

Рис. 1. Алгоритм автоматичної генерації коду OpenCL

Далі, згідно з розробленою методикою, слід оцінити можливу загальну ефективність виконання ділянки програмного коду на GPU [4]. Для об'єктивної оцінки ефективності обчислень на графічному процесорі крім безпосередньо часу виконання фрагменту коду на GPU ($T_{GPU\text{вик}}$) слід враховувати додатковий час, який витрачається на створення контексту виконання програми, виділення ресурсів графічного адаптеру, копіювання даних з основної пам'яті у пам'ять відеокарти, копіювання результату та вивільнення ресурсів GPU ($T_{GPU\text{додат}}$). Отже, доцільно виконувати код на GPU тільки тоді, коли

$$T_{GPU\text{заг}} \leq T_{CPU\text{вик}},$$

де $T_{GPU\text{заг}} = T_{GPU\text{вик}} + T_{GPU\text{додат}}$

Розглянемо ряд базових алгоритмів та перевіримо доцільність їх виконання на графічних процесорах для даних різної розмірності. Нехай коефіцієнт *K*– відношення кількості операцій, які виконуються на GPU до кількості пересилок даних між CPU та GPU.

1.Лінійні матрично-векторні алгоритми.

1.1. Знаходження суми векторів. Нехай необхідно знайти поелементну суму двох векторів розмірності *n*. Маємо *n* операцій (додавань) та *3n* переміщень між CPU та GPU. Коефіцієнт*K*– 1:3 або $\theta(1)$. Отже, виграш від виконання такого коду на GPU буде дуже незначним або взагалі відсутнім.

1.2. Знаходження добутку матриць. Для знаходження добутку двох матриць розмірності $n*n$ слід виконати n^3 операцій (множень та додавань) та $3n^2$ переміщень. Коефіцієнт K – $\theta(n)$. Таким чином, виконання такого коду на GPU буде набагато швидше ніж на CPU і виграш стрімко зростатиме із збільшенням розмірності масиву.

2. Паралельна редукція суми. Редукцією масиву a_0, a_1, a_2, …, a_{n-1} відносно заданої операції (суми) буде наступна величина A = $(((a_0 + a_1) + a_2) + \ldots + a_{n-1})$. Коефіцієнт *K*– 1:2 або $\theta(1)$, а отже виграшу у швидкодії не буде.

3. Двійковий пошук. Для знаходження елементу масиву розмірності *n* за допомогою двійкового пошуку необхідно здійснити $\log_2 n$ операцій та *2n* переміщень. У такому випадку виконання коду на GPU є нераціональним.

4. Порозрядне сортування (radix sort). Часова складність порозрядного сортування для масиву розмірності *n* становить $\theta(pn)$, де *p* – додатковий об'єм пам'яті необхідний для сортування ($p < n$). Кількість переміщень між CPU та GPU – *2n*. Отже, виграш у швидкодії буде помітним, але не таким значним як для випадку 1.2.

Практичні результати дослідів представлені на рис. 2. Використане апаратне забезпечення – комп'ютер з процесором Intel Core 2 Duo T5850 (2,16 ГГц) та графічним адаптером NVIDIA GeForce 8600M GS.

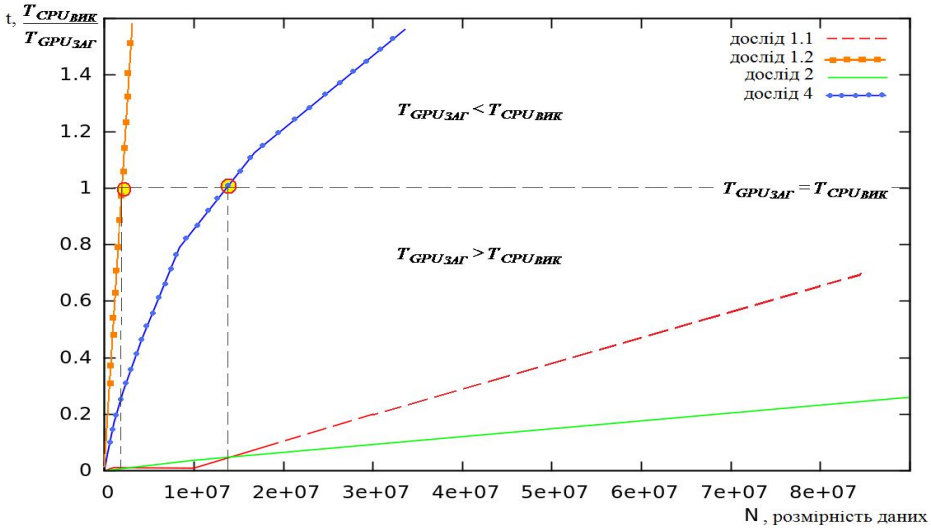

Рис. 2. Експериментальні дані швидкодії програмних алгоритмів

Отримані графіки відображають відношення загального часу виконання програми на GPU до часу її виконання на CPU для даних різної розмірності. Очевидно, що на графічному процесорі серед розглянутих операцій найдоцільніше виконувати перемноження матриць (дослід 1.2) та порозрядне сортування (дослід 4).

Отже, у ході роботи експериментальним способом було доведено, що на графічних процесорах раціонально виконувати програмний код у таких випадках:

1. Для масивно-паралельних обчислень над великими обсягами даних, для яких складність виконуваних на GPU операцій нівелює вартість пересилок даних між CPU та GPU. Теоретична перевірка цього твердження зводиться до розрахунку відношення кількості операцій до кількості переміщень. Якщо порядок цього відношення залежить від розмірності даних, то швидкодія виконання програмного коду на графічному процесорі у порівнянні із звичайним процесором стрімко зростатиме при збільшенні розмірності даних.

2. Оскільки кількість пересилок між центральним та графічним процесорами повинна бути мінімальною, то дані слід зберігати в пам'яті GPU якнайдовше та послідовно виконувати над ними декілька операцій. В подальшому було б корисно дослідити цей підхід з метою досягнення максимальної ефективності функціонування гетерогенної системи.

Якщо оцінка загальної ефективності виконання програмної ділянки на графічному процесорі показала, що генерація OpenCL коду є доцільною, то можна переходити до наступного етапу – побудови графових моделей програми.

Деякі цикли не можуть бути перетвореними для виконання на графічному процесорі. Можливість розпаралелення циклу визначається залежностями між даними. Так, наприклад, існують цикли, які не можна ні розпаралелити, ні перетворити жодним чином. Відповідно, виконати такі цикли на процесорах SIMD-архітектури та отримати коректний результат неможливо. Розроблені засоби автоматичної генерації коду здатні визначати залежності між даними у тілі циклу. Для циклів із залежностями генерується послідовний код, який виконується на центральному процесорі.

На цьому етапі необхідно створити граф інформаційних зв'язків програми. На його основі аналізуються залежності у програмі та приймається рішення про можливість або неможливість виконання ділянки коду на графічному процесорі.

Якщо в циклі немає залежностей, то ітерації можна виконувати паралельно на будь-яких архітектурах. Асинхронна архітектура (багатоядерні процесори, кластери) дозволяє розпаралелювати цикли будь-якої глибини вкладеності. Ці цикли можуть містити умовні оператори, але допускають тільки циклічно незалежні залежності. Синхронна архітектура дозволяє розпаралелювати найбільш глибоко вкладені цикли і допускає залежності, які спрямовані зверху вниз. Спільна пам'ять допускає вхідні залежності. При розподіленій пам'яті вхідна залежність заважає, але її можна усунути попереднім копіюванням даних.

Побудова графу інформаційних зв'язків [5]. Нехай X – змінна в програмі. Входженням змінної X будемо називати будь-яку появу цієї змінної в тексті програми. Входженню змінної при конкретному значенні індексного виразу відповідає звернення до деякої комірки пам'яті. Якщо при цьому зверненні змінюється стан комірки (входження в ліву частину оператора присвоювання, яке не входить в індексний вираз іншого входження), то таке входження називається генератором. Інші входження називаються використаннями.

Нехай входження *v* залежить від входження *u*. Розрізняють чотири типи інформаційних залежностей:

1. Якщо *u* є генератором, а *v* – використанням, то залежність такого типу називається істинною (true dependence) або потоковою залежністю (flow dependence);
2. Якщо *u* є використанням, а *v* – генератором, то залежність такого типу називається антизалежністю (anti dependence);
3. Якщо *u* і *v* є генераторами, то залежність такого типу називається вихідною залежністю (output dependence) або залежністю по виходу;
4. Якщо *u*і *v*є генераторами, то залежність такого типу називається вхідною залежністю (input dependence) або залежністю по входу.

Якщо входження *v* залежить від себе ж (входження *v*), то будемо говорити, що має місце самозалежність входження *v*.

Розглянемо фрагмент програми множення двох многочленів:

```
for (k = 0; k <= (N + M); k + +)
c [k] = 0;(u1)
for (i = 0; i<= M; i + +)
for (j = 0; j <= N; j + +)
c [i + j](u2)= c [i + j] (v1)+ a [i] (v2)* b [j](v3).
```

Перерахуємо всі інформаційні залежності в даному прикладі.

1. Потокові залежності: залежність використання *v1*від генератора *u1*,і залежність використання *v1*від генератора *u2*.
2. Вихідні залежності: залежність генератора *u2*від генератора *u1*, і залежність генератора *u2*від самого себе – самозалежність.
3. Антизалежність: залежність генератора *u2* від використання *v1*.

4. Вхідна залежність: самозалежність використання *v1*, самозалежність використання *v2*, самозалежність використання *v3*.

Особливим чином повинен будуватися граф залежності фрагмента програми, який містить виклики функцій. Операція (оператор) виклику функції розглядається як одна вершина графа. Якщо після заміни формальних змінних фактичними, у тілі функції є генератор (застосування) деякої змінної *X*, то з даної вершини (у дану вершину) графа йдуть дуги, як з генератора (застосування) *X*. Для більшої точності, виклик процедури слід тимчасово замінити відповідним текстом підпрограми, побудувати граф залежностей для даного фрагмента з цим текстом, а потім увесь підграф, який відповідає вставленому тексту, замінити на одну вершину.

При розробці програмного комплексу автоматичної генерації коду OpenCL розпаралелювались цикли, які не мають інформаційних залежностей та такі, які мають вхідні залежності. В результаті була отримана програмна система, яка дозволяє генерувати ефективний код OpenCL та виконувати його на графічному процесорі гетерогенної системи. Користувацький інтерфейс системи представлений на рис. 3.

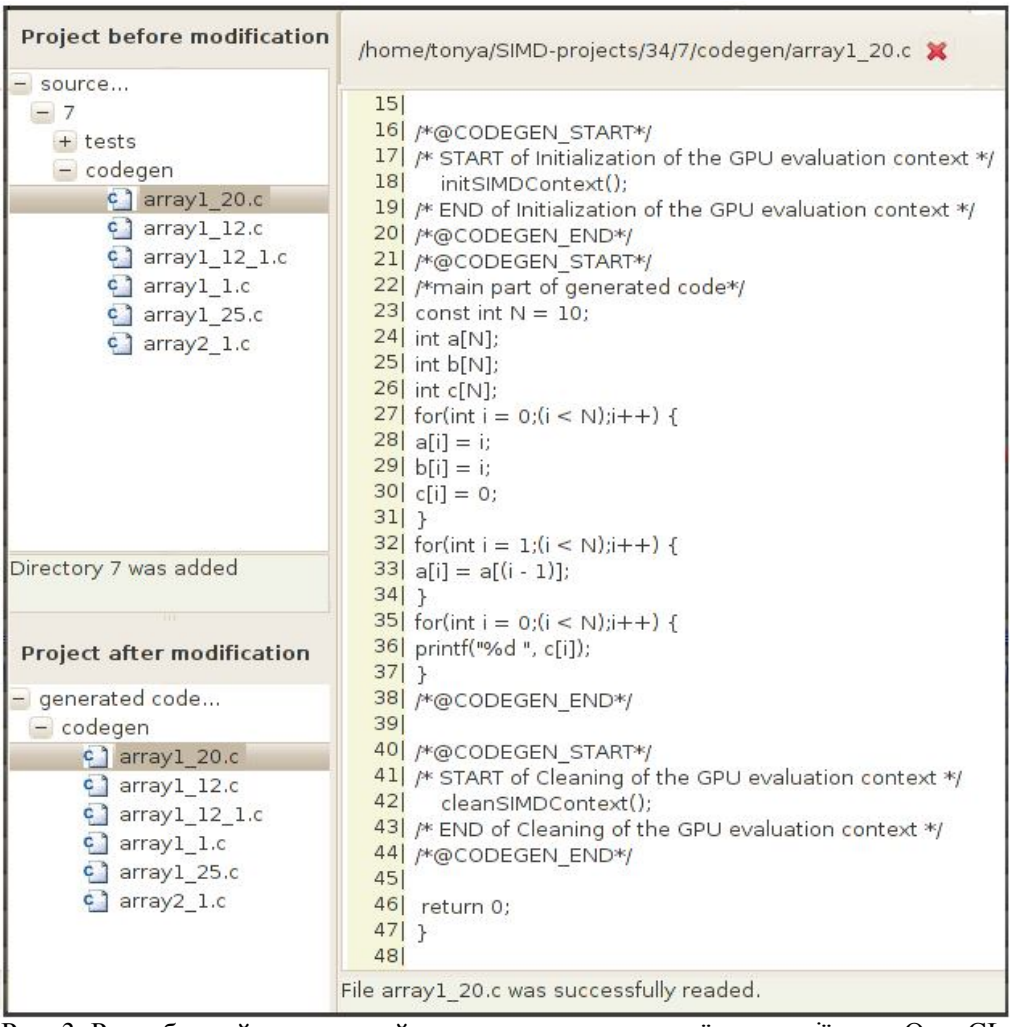

Рис. 3. Розроблений програмний комплекс автоматичної генерації коду OpenCL

Висновки. У роботі вперше запропонована оригінальна методика, яка дозволяє автоматично генерувати ефективний програмний код OpenCL. Згенерована програма може бути виконана у гетерогенній комп'ютерній системі і при цьому дозволить отримати приріст швидкодії системи в цілому та збереження семантичної коректності вихідного коду.

На основі цієї методики був розроблений програмний комплекс, який визначає ділянки коду, які можливо та доцільно виконати на GPU, генерує функції-ядра графічного процесора та порівнює отримані таким чином результати зі звичайним послідовним виконанням коду на CPU.

У подальшому доцільно додатково дослідити можливості більш ефективного використання пам'яті GPU та враховувати їх при копіюванні даних між різними типами пристроїв системи.

Література

1.Benedict R. Gaster.The OpenCL C++ Wrapper API, Version 1.1 / Benedict R. –Gaster Khronos OpenCL Working Group, June 14, 2010. –101 с.

2. NVIDIA, OpenCL Best Practices Guide, May 27, 2010.–53с.

3.Уточнение зависимостей программы в ДВОР / [Б.Я. Штейнберг, А.А. Абрамов, А.П. Баглийи др.] // Труды международной конференции «Параллельные вычисления и задачи управления».– Москва, 26-28 октября 2010.– С. 855-864.

4. Замятін Д.С. Оцінка швидкодії базових алгоритмічних задач в гетерогенних комп'ютерних системах / Д.С. Замятін, А.К. Чередніченко // Збірник тез конференції ПМК2011, Київ, НТУУ «КПІ». – С. 35.

5. Воеводин В.В. Параллельные вычисления / В.В. Воеводин, Вл.В. Воеводин. – Санкт-Петербург «БХВ-Петербург», 2002. – 608 с.

УДК 621.391

Смелянский А.А., асп. *(Гос. университет информационно-коммуникационных технологий)*

РАСПРОСТРАНЕНИЕ РАДИОСИГНАЛА ЧЕРЕЗ ПРЕПЯТСТВИЕ

Смілянський А.О. Поширення радіосигналу через перешкоду. Виконано розрахунок проходження радіосигналу всередині приміщення, з врахуванням відбиття і заломлення падаючої хвилі при попаданні на перешкоду, дано наочне представлення процесу проходження та відбивання.

Ключові слова: РАДІОХВИЛЯ, РАДІОСИГНАЛ, ВІДНОСНА ДІЕЛЕКТРИЧНА ПРОНИКНІСТЬ

Смелянский А.А. Распространение радиосигнала через препятствие. Выполнен расчет прохождения радиосигнала внутри помещения, с учетом отражения и преломления падающей волны при попадании на препятствие, дано наглядное представления процесса прохождения и отражения.

Ключевые слова: РАДИОВОЛНА, РАДИОСИГНАЛ, ОТНОСИТЕЛЬНАЯ ДИЭЛЕКТРИЧЕСКАЯ ПРОНИЦАЕМОСТЬ

Smilianskyi A.O. Radio propagation over a hurdle. Calculation of a radio signal indoors, with taking into account of the reflection and refraction of the incident wave in contact with an obstacle, a visual representation of the process of transmission and reflection.

Keywords: RADIO WAVES, RADIO SIGNAL , RELATIVE DIELECTRIC CONSTANT

Проблеме распространения радиоволн внутри зданий и помещений последнее время уделяется большое внимание. Это связано, прежде всего, с созданием локальных информационных сетей, а также с необходимостью обеспечения надежной радиосвязью сотрудников предприятий, учреждений с целью оперативного управления и обеспечения безопасности. Наличие внутри здания стен, перегородок, мебели, радиоэлектронной аппаратуры, людей и других объектов создает сложную среду распространения радиоволн. Условия распространения радиоволн внутри помещений существенно отличаются от условий распространения радиоволн в свободном пространстве. Основными эффектами, наблюдаемыми при распространении радиоволн внутри помещений, являются многолучевость, обусловленная многократными отражениями радиоволн от стен и других объектов, дифракция на многочисленных острых кромках предметов, расположенных внутри комнаты, и рассеяние радиоволн. Эти эффекты создают сложную интерференционную структуру электромагнитного поля, сильно изменяющуюся при перемещении людей и других объектов [1…3].