УДК 681.3;004.93

А. А. ЯРОВИЙ[1], О. О. КУЛИК[1], Н. І. КОКРЯЦЬКА[2]

[1] Вінницький національний технічний університет, Вінниця
[2] Державний економіко-технологічний університет транспорту, Київ

ПАРАЛЕЛЬНО-ІЄРАРХІЧНЕ ПЕРЕТВОРЕННЯ ПЛЯМОПОДІБНИХ ЗОБРАЖЕНЬ НА ОСНОВІ MULTI-GPU СИСТЕМ

Анотація. В проведених дослідженнях здійснено аналіз особливостей організації високопродуктивних паралельно-ієрархічних обчислювальних процесів на базі Multi-GPU систем та технологій. Розроблено програмний комплекс призначений для реалізації прямого паралельно-ієрархічного перетворення плямоподібних зображень з використанням обчислювальних потужностей двох графічних карт, які об'єднані у єдину систему. Досягнено підвищення швидкодії класифікації зображень профілю лазерного променя в паралельно-ієрархічних мережах на основі Multi-GPU систем.
Ключові слова: паралельні обчислення, паралельно-ієрархічне перетворення, обробка зображень, GPGPU, CUDA, OpenMP, Multi-GPU Programming.
Аннотация. В проведенных исследованиях осуществлен анализ особенностей организации высокопродуктивных параллельно-иерархических вычислительных процессов на базе Multi-GPU систем и технологий. Разработан программный комплекс, предназначенный для реализации прямого параллельно-иерархического преобразования пятноподобных изображений с использованием вычислительных мощностей двух объединённых в единую систему графических карт. Получен прирост быстродействия классификации изображений профиля лазерного луча в параллельно-иерархических сетях на базе Multi-GPU систем.
Ключевые слова: параллельные вычисления, параллельно-иерархическое преобразование, обработка изображений, GPGPU, CUDA, OpenMP, Multi-GPU Programming.
Abstract. The analysis of peculiarities of organization of highly productive parallel-hierarchical computing processes on the base of Multi-GPU systems and technologies is carry out in researches. The software package for realization of parallel-hierarchical transformation of spot images using computational capability of two graphics card integrated in joint system is developed. The performance increase of classification of laser beam profile in parallel-hierarchical networks on the base of Multi-GPU systems is obtained.
Key words: parallel computing, parallel-hierarchical transformation, image processing, GPGPU, CUDA, OpenMP, Multi-GPU Programming.

Вступ

Останнім часом спостерігається тенденція до зростання потреби у проведенні ресурсоемних обчислень, аналізу та обробки великих обсягів інформації. До задач такого типу можливо віднести, наприклад, задачі прогнозування погоди та змін клімату, моделювання складних фізичних процесів, розпізнавання зображень, синтез мови людини, та багато інших. Такий стан речей в свою чергу призводить до підвищення актуальності методів, що здатні використовувати усі переваги концепції паралелізму. Паралельно-ієрархічне перетворення належить саме до групи таких методів.

Паралельно-ієрархічне (ПІ) оброблення зображень знаходить застосування у вирішенні ресурсоємних та складних задач у багатьох прикладних галузях, наприклад, у медицині, астрополяриметрії, тощо. Оскільки, в таких задачах обробку доводиться виконувати у реальному часі, виникає потреба у підвищенні швидкодії. Саме такі чинники зумовлюють актуальність використання GPU-систем для реалізації ПІ перетворення [1].

З урахуванням обмеженої продуктивності одного окремо взятого графічного процесора найбільший інтерес для досліджень становить реалізація паралельних алгоритмів, зокрема ПІ перетворення, на основі системи, що містить декілька графічних карт, або навіть на основі кластерного об'єднання GPU-орієнтованих систем. Варто відзначити, що дослідження в даному напрямі є актуальними, про що свідчать останні дослідження провідних світових компаній, наприклад, NVIDIA Corporation. Окрім того, наразі є наявною велика кількість засобів для досягнення паралельності: починаючи від GPGPU-платформ (NVDIA CUDA, AMD FireStream, OpenSL) закінчуючи відкритими стандартами та бібліотеками для певних мов програмування (OpenMP, Cilk, TBB та інші). Велика кількість альтернатив дозволяє обирати найефективніший засіб для кожної конкретної задачі, або навіть об'єднувати їх з метою досягнення ще більшого рівня розпаралелювання і, відповідно, ще більшого приросту швидкодії [1].

Публікація містить результати досліджень, проведених при грантовій підтримці Державного фонду фундаментальних досліджень за конкурсним проектом Ф61/199-2015 "Методологія побудови високопродуктивних інтелектуалізованих паралельно-ієрархічних систем на основі сучасних мережевих обчислювальних комплексів з гетерогенною архітектурою".

Мета дослідження

Метою дослідження є підвищення швидкодії класифікації зображень профілю лазерного променя в паралельно-ієрархічних мережах на основі Multi-GPU систем.

Задачі дослідження

Відповідно до мети підлягають вирішенню такі задачі:

– аналіз прямого ПІ перетворення в контексті його реалізації на основі GPGPU;

– аналіз сучасних платформ для реалізації GPGPU-обчислення з урахуванням підтримки одночасного використання декількох графічних процесорів;
– реалізація прямого ПІ перетворення на основі Multi-GPU системи.

Опис процесу організації обчислень у паралельно-ієрархічному перетворенні

Принцип паралельно-ієрархічного оброблення інформації припускає організацію багаторівневого ПІ обчислювального процесу, орієнтованого на досягнення максимально можливої алгоритмічної та схемотехнічної швидкодії при перетворенні інформації. ПІ перетворення застосовується для виділення характерних ознак зображень, їх кодування і скорочення розмірності при виконанні обчислень. Добра збіжність ПІ перетворення використовується в структурах паралельної пам'яті, аналізу і розпізнавання зображень, при кодуванні і ущільненні даних, а також для обробки біомедичних сигналів. Особливо перспективною є ідея реалізації ПІ перетворення при побудові оптоелектронних елементів і пристроїв із динамічною багатофункціональністю [1-3].

Мережний метод паралельно-ієрархічного перетворення, в тому числі на рівні моделей вже був розглянутий в багатьох роботах [1-3]. Але для кращого подальшого розуміння організації обчислювального процесу, наведемо основні складові математичної моделі. Нехай є S ($S=1,2,3,\ldots$) непустих множин елементів, що задають інформацію. Кількість елементів множини є її довжиною, тобто L_μ – довжина множини μ. Кількість різноманітних елементів множини є розмірністю даної множини (R_μ). Математична модель пірамідального розкладання множини $\mu = \{ a_i \}$, $i = \overline{1, n}$ має вигляд [2]:

$$\sum_{i=1}^{n} a_i = \sum_{j=1}^{R} \left(n - \sum_{k=0}^{j-1} n_k \right) \left(a^j - a^{j-1} \right), \tag{1}$$

де R – розмірність даної множини; $a_i \neq 0$. З однакових елементів сформуємо підмножини, елементи однієї підмножини позначимо через a^k, $k = \overline{1,R}$, відповідно n_k – кількість елементів у k-тій підмножині (тобто кратність числа a_k), a^j – довільний елемент множини $\{a^k\}$, обраний на j-му кроці, $j = \overline{1,R}$, $a^0 = 0$, $n_k = 0$. Аналізуючи вираз пірамідальної обробки числової інформації (1) можна зробити висновок про те, що у процесі обробки з кожним кроком кількість чисел зменшується. Якщо множини одержувані після кожного кроку поставити послідовно одну на іншу, то утворений ними тривимірний контур буде мати форму піраміди. Перетворенням множини μ в множину μ^1, що задається моделлю (1), є оператор перетворення G [2]:

$$G(\mu) = \mu^1. \tag{2}$$

Якщо для вихідних S масивів (S – непусті множини елементів, що задають інформацію) застосуємо оператор перетворення G, що задається виразом (1), то для кожного масиву отримуємо свій порядковий розклад [2]:

$$\mu_1^1 = \bigcup_{i=1}^{R_1^1} a_{1i}^i \,, \quad \mu_2^1 = \bigcup_{i=1}^{R_2^1} a_{2i}^i \,, \quad \ldots \,, \quad \mu_s^1 = \bigcup_{i=1}^{R_s^1} a_{si}^i \,, \tag{3}$$

де μ_s^1 – множина з номером S на першому рівні, тоді для k-го рівня множина з номером l записується μ_l^k, відповідно R_s^1 – кількість елементів у S множині на першому рівні, R_l^k – кількість елементів у S множині на k-му рівні.

Мережний метод прямого паралельно-ієрархічного перетворення полягає в послідовному застосуванні до початкових множин $\bigcup_{s=1}^{S} \mu_S$ по одному разу операторів перетворення G і транспонування T, а потім $(k-1)$ раз функціонала Φ [2]:

$$\underset{t=2}{\overset{k}{\Phi}}\left[T\left(G\left(\bigcup_{s=1}^{S}\left(\bigcup_{i=1}^{n} a_i\right)\right)\right)\right] = \bigcup_{t=2}^{k} a_{11}^{t}, \quad (4)$$

де a_{11}^{k} – вихідна інформація (хвостові елементи мережі) прямого паралельно-ієрархічного перетворення.

Тобто, послідовне застосування трьох операторів *G, S, T* формує функціонал *Ф*, тобто $\Phi(M) = T\left[S\left(G\left(M\right)\right)\right]$, де *S* – оператор зсуву рядка на величину меншу номера даного рядка на одиницю і виключення першого стовпчика матриці M_k в результат розкладу [1-3].

З практичної точки зору алгоритм прямого ПІ перетворення складається з виконання трьох фіксованих операції у певному порядку: транспонування, зсув та G-перетворення. На вхід алгоритму подається матриця даних. В ході обробки за допомогою вищенаведених операцій знаходиться матриця хвостових елементів, яка і є результатом роботи алгоритму. Варто відзначити, що виконання кожної з цих операції добре підлягає паралельній реалізації, як на CPU-, так і на GPU-орієнтованій апаратній платформі. Більш того, оскільки за допомогою ПІ перетворення, як правило, обробляють великий набір зображень (десятки тисяч), а обробка кожного окремного зображення не залежить від результатів обробки інших зображень, то немає фактично жодних перешкод для того, щоб реалізувати обробку зображень в декілька потоків, наприклад, на декількох графічних картах.

Паралельно-ієрархічне перетворення в контексті технології GPGPU

Останнім часом в наукових колах все ширше обговорюється тема високопродуктивних обчислень і суперкомп'ютерів. Вони непогано освоєні, для них розроблено велику кількість прикладних пакетів, з їх допомогою вирішуються складні обчислювальні задачі. Вони мають багато переваг, за винятком таких показників, як висока вартість та споживання електроенергії, що змушує шукати альтернативу. Однією з них є GPGPU – технологія використання графічного процесора відеокарти для виконання загальних обчислень. Однією з головних переваг GPU є велика кількість ядер (близько 2500 в сучасних відеокартах), що дозволяє в повній мірі реалізувати переваги паралелізму. Теоретична продуктивність сучасних відеокарт вже набагато вища, ніж у відповідних CPU (рис.1) [4].

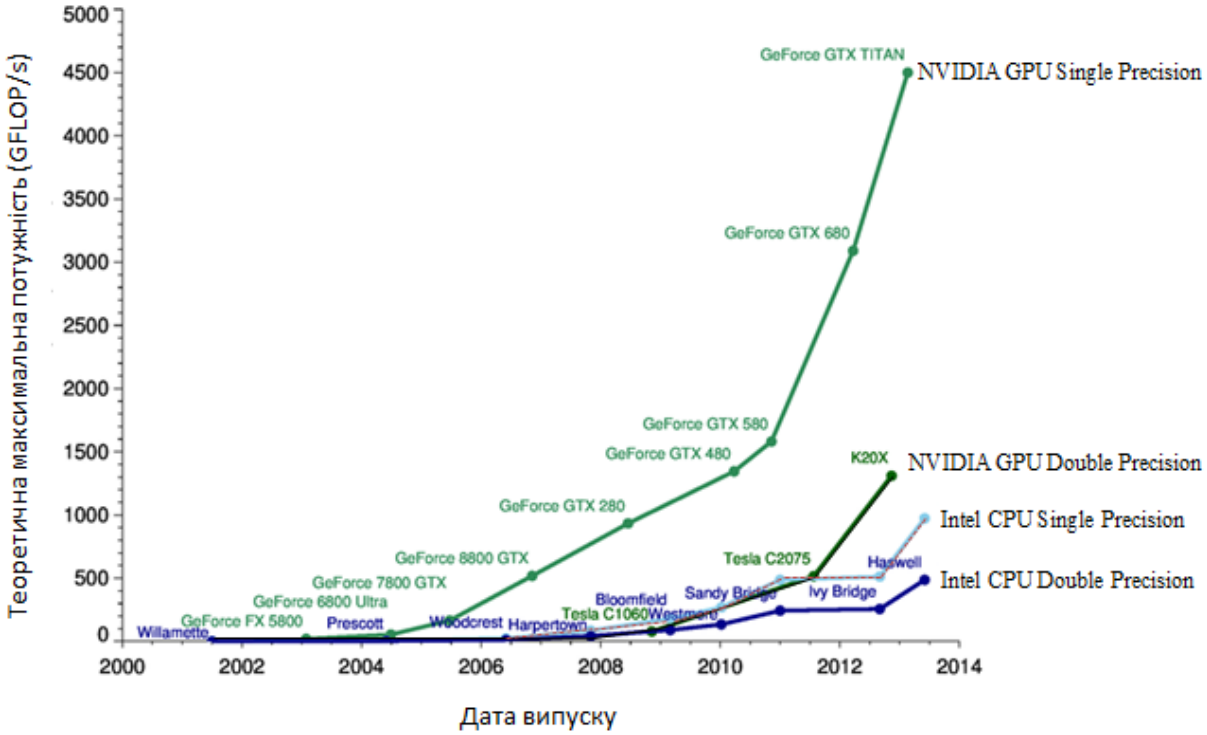

Рисунок 1 – Порівняння продуктивності CPU- та GPU-орієнтованої апаратної платформи

Природно, далеко не на всіх задачах вдається отримати високу продуктивність навіть на одному GPU. Тут можна виділити спільні системні вимоги для задач, що потребують реалізації обчислень загального призначення на GPU:

1. Висока обчислювальна продуктивність.
2. Велика розмірність обчислюваного масиву.
3. Можливість незалежного обчислення цільових елементів на кожному з "проходів".
4. Масиви даних, що використовуються при обчисленнях, повинні повністю поміщатися у відеопам'ять.

Розглянемо головні особливості структури та принципів функціонування GPGPU-програм. В якості прикладу архітектури доцільно використати технологію NVIDIA CUDA, як найбільш розвинену на даний момент. Характерною особливістю будь-якої GPGPU-програми є використання графічної карти для проведення обчислень. В термінології CUDA центральних процесор має назву HOST, а графічний процесор – DEVICE. Варто відзначити те, що GPU, фактично, є периферійним пристроєм, який отримує всі команди з CPU. Таким чином, типова структура GPGPU-програми має такий вигляд [5,6]:

1. CPU виділяє місце в пам’яті GPU для подальшого копіювання інформації.
2. CPU копіює інформацію в пам’ять GPU.
3. CPU запускає паралельні підпрограми на GPU для обробки інформації.
4. CPU копіює оброблені дані назад з GPU.
5. CPU звільняє місце в пам’яті GPU.

З точки зору програміста важливим є те, що програмний код для виконання на GPU описується в окремій функції, яка має назву kernel-функції, і може буде викликана з будь-якої позиції, що значно підвищує гнучкість програми в цілому. Варто відзначити, що kernel-функція виконується для кожного зі створених потоків одночасно.

Натепер технологія GPGPU реалізована в різноманітних програмно-апаратних архітектурах. Найбільш поширеними з них є NVIDIA CUDA, OpenCL та AMD FireStream. Коротко розглянемо переваги і недоліки кожної з них.

CUDA (Compute Unified Device Architecture) – створена NVIDIA платформа для паралельних обчислень, а також модель прикладного програмного інтерфейсу (англ. API). Вона дозволяє розробникам використовувати графічний процесор для проведення неграфічних обчислень. Окрім того, CUDA надає широкий спектр засобів для безпосереднього налаштування роботи GPU та реалізації паралелізму.

Інтерфейс CUDA заснований на стандартній мові програмування C, що дозволяє спростити використання ресурсів GPU, оскільки розробнику не потрібно мати великий досвід в графічному програмуванні, на відміну від таких API, як, наприклад, Direct3D та OpenGL, які цього потребують.

CUDA забезпечує як API низького рівня (CUDA Driver API), так і більш високого (CUDA Runtime API). Перша версія CUDA SDK була оприлюднена 15 лютого 2007 року, для Microsoft Windows і Linux. Підтримка Mac OS X була додана пізніше у версії 2.0, яка з’явилася 14 лютого 2008. CUDA працює з усіма графічними процесорами Nvidia починаючи з серії G8x, включаючи серії GeForce, Quadro та Tesla [5,6].

В порівнянні з аналогами CUDA має велику кількість переваг [5,6]:

– інтеграція в єдиний простір з іншими технологіями від NVIDIA, наприклад, NVIDIA SLI;
– підтримка Multi-GPU Programming;
– високий рівень інноватівності в порівнянні з аналогами;
– порівняно висока ефективність транзакцій між пам’яттю CPU та GPU;
– наявність розподіленої пам’яті (shared memory), що дозволяє значно пришвидшити роботу з пам’яттю;
– наявність єдиної віртуальної пам’яті (починаючи з CUDA 4.0) та єдиної пам’яті (починаючи з CUDA 6.0);

Серед недоліків CUDA варто відзначити такі:

– CUDA підтримує тільки відеокарти від NVIDIA;
– CUDA не підтримує стандарт C в повній мірі, що в свою чергу накладає певні обмеження.

ATI FireStream – повний аналог технології CUDA від NVIDIA. Він надає можливість використання шейдерів графічного процесора для запуску обчислювальних програм. Інтерфейс програмування здійснюється через OpenCL. Це дає можливість прискорення обчислень, і може бути використано, зокрема, в ігровій сфері, для прискорення прорахунків фізики, якщо програмне ядро фізики підтримує OpenCL. Варто відзначити, що FireStream підтримує лише графічні карти від ATI. В цілому, дана технологія значно поступається NVIDIA CUDA за своїм функціоналом та придатністю до виконання гетерогенних обчислень.

OpenCL (Open Computing Language) – це фреймворк для написання програм, що виконуються на гетерогенних платформах, які можуть містити центральні процесор (CPU), графічні процесори (GPU), ци-

фрові сигнальні процесори (DSP), програмовані користувачем вентильні матриці (FPGA) та інші процесори [7].

OpenCL визначає мову (на основі C99) для програмування цих пристроїв та інтерфейси прикладного програмування (API) для управління платформою і виконання програм на обчислювальних пристроях. Даний фреймворк забезпечує паралельні обчислення, використовуючи паралелізм на основі завдань і даних. OpenCL є відкритим стандартом, що підтримується некомерційним технологічним консорціумом KhronosGroup [7].

OpenCL розглядає обчислювальну систему як таку, що складається з ряду обчислювальних пристроїв, якими можуть бути центральні процесори (CPU) або "прискорювачі", такі як графічні процесори (GPU), прикріплені до хост-процесору (CPU). Він визначає C-подібну мову для написання програм, так званих ядер (kernels), які виконуються на обчислювальних пристроях. Один обчислювальний прилад, як правило, складається з багатьох окремих елементів обробки (PEs) і виконання одного ядра (kernel) може проводитися на всіх або багатьох PEs паралельно.

Ключовою особливістю OpenCL є портативність, яка досягається за допомогою використання абстрактної пам'яті та виконання моделі, при якій програміст не в змозі безпосередньо використовувати апаратнозалежні технології, такі як ParallelThreadExecution (PTX) для графічних процесорів NVIDIA, якщо він не готовий відмовитися від прямого перенесення на інші платформи. При виконанні цих умов можна запускати будь-яке ядро (kernel) OpenCL на будь-якій сумісній реалізації [7].

На основі вищенаведеного можна зробити висновок, що для реалізації алгоритму ПІ перетворення найбільш придатною є програмна платформа NVIDIA CUDA, оскільки має в своєму розпорядженні більшу кількість засобів для реалізації паралельних обчислень. Окрім того, деякі засоби, як, наприклад, Multi GPU-Programming, не мають прямих аналогів на інших програмних платформах і значно спрощують виконання обчислень на декількох відеокартах.

Паралельно-ієрархічне перетворення перетворення за допомогою NVIDIA CUDA

Ідея реалізації алгоритму прямого ПІ перетворення на основі GPGPU технологій не є принципово новою і вже була розглянута у ряді робіт [8-11]. Одним з найбільш перспективних напрямів реалізації є використання технології NVIDIA CUDA. Однією з переваг CUDA є наявність додаткових бібліотек, які дозволяють значно спростити паралельне програмування на GPU. Прикладом такої бібліотеки є CUDA Thrust. Thrust – реалізована на мові програмування C++ бібліотека шаблонів для CUDA, що основана на STL (Standard TemplateLibrary). Вона дозволяє використовувати при GPU - обчисленнях такі класи STL як "vector" чи <list>. Окрім того, Thrust дозволяє використовувати адаптовані під виконання на GPU версії таких алгоритмів, як "reduce", "sort", "copy" та інші [12]. Такі можливості значно спрощують реалізацію ПІ перетворення на основі NVIDIA CUDA і дозволяють зробити кінцевий програмний засіб набагато більш гнучким. Так, реалізація прямого ПІ перетворення на основі NVIDIA CUDA з використанням CUDA Thrust забезпечує середнє збільшення швидкодії в 1,4 рази в порівнянні з реалізацією на CPU при роботі з набором плямоподібних зображень формату ".bmp" та розмірністю 2048×2048 пікселів. Варто відзначити, що використання GPGPU-технологій зазвичай виправдовує себе лише при роботі з зображеннями великої та надвеликої розмірності, під якими, як правило, розуміють зображення з розмірністю від 1024×1024 пікселя та більше. При цьому спостерігається наступна тенденція: чим більше розмірність зображення, тим більший приріст швидкодії отримується при використанні GPGPU-технологій [8,9]. Такі результати підтверджують доцільність використання обраних технологій в контексті ПІ оброблення зображень.

Після реалізації ПІ перетворення на основі одного GPU та підтвердження раціональності такого підходу наступним логічним кроком стає оцінка доцільності реалізації алгоритму, що буде використовувати декілька графічних карт. Ця задача значно спрощується тим, що платформа NVIDIA CUDA вже містись в собі засоби для реалізації подібних обчислень, а саме концепцію Multi-GPU Programming.

Multi-GPU Programming – набір засобів, спрямованих на одночасне використання декількох графічних карт при виконанні CUDA-обчислень. Такий підхід забезпечує підвищення швидкодії та, з певними обмеженнями, збільшення розміру доступної для обчислень пам'яті. Окрім того, варто відзначити високу масштабованість даної технології. За потреби, обчислювальна потужність Multi-GPU системи може бути збільшена шляхом підключення додаткових графічних карт, які можливо розмістити як на одному комп'ютері, так і на декількох, об'єднаних в одну мережу [10,11].

Можливо виділити два варіанти реалізації технології Multi-GPU Programming:

– на основі окремного комп'ютера з декількома GPU,

– на основі мережі з декількох комп'ютерів.

Кожен з цих підходів має свої переваги та недоліки, але головним критерієм при виборі все ж залишається наявність потрібних апаратних засобів та відповідного програмного забезпечення. Розглянемо кожен з цих варіантів.

Спосіб реалізації Multi-GPU Programming на одному комп’ютері залежить від того, чи потрібна взаємодія між графічними картами під час виконання програми. Найпростішим є варіант без взаємодії, тобто коли кожна графічна карта працює незалежно від інших. Оскільки всі функції та події (events) CUDA виконуються відносно поточного GPU, а зміна поточного GPU на інший не призводить до зупинки роботи попереднього, то для забезпечення роботи програми достатньо викликати потрібні kernel-функції для кожного GPU. Вибір поточного GPU здійснюється за допомогою функції cudaSetDevice(), параметром якої є номер потрібного GPU.

Для реалізації прямого обміну даними між графічними картами використовується функція cudaMemcpyPeerAsync(), яка приймає в якості параметрів: номера графічних карт, покажчики на адреси в пам’яті з яких і в які потрібно копіювати та розмір в байтах. Якщо є в наявності P2P (peer-to-peer) з’єднання, то дані копіюються напряму через PCI шину. В іншому ж випадку використовується проміжна пам'ять на CPU.

Для реалізації Multi-GPU Programming на декількох комп’ютерах потрібно спочатку встановити з’єднання між ними, наприклад, за допомогою MPI. Також, на кожному комп’ютері повинна бути встановлена щонайменш одна графічна карта від NVIDIA. Варто відзначити те, що у випадку, коли окремий комп’ютер обладнаний декількома графічними картами, вони також можуть використовуватися одночасно. Подальші дії залежать від того, чи потрібна взаємодія між графічними картами під час виконання CUDA-програми. Якщо ні, то завдання розподіляються так само, як і у звичайній мережі, за винятком того, що вони містять GPGPU-код, а не звичайний CPU.

Випадок, коли потрібна взаємодія між графічними картами є більш складним. Наразі NVIDIA CUDA не надає можливості прямого обміну даними між відеокартами, що знаходяться на різних комп’ютерах. Тому якщо виникає потреба в такому обміні, то він здійснюється таким чином:

– копіювання з GPU на CPU,

– обмін даними між CPU у мережі,

– копіювання з CPU на GPU.

Такий спосіб є досить трудомістким і його використання не завжди доцільно. В даному дослідженні головну увагу приділено, зважаючи на наявні програмні та апаратні ресурси, саме першому варіанту, а саме обчисленням в межах однієї комп’ютерної системи з двома графічними картами. Але оскільки обчислювальне навантаження може бути розподілене між двома графічними картами багатьма різними способами, постає питання про вибір найбільш доцільного підходу. У контексті задачі ПІ оброблення зображень можливо виділити два принципово відмінних варіанти: обробка кожного окремого зображення за допомогою двох графічних карт шляхом розподілення навантаження між ними або виділення кожній графічний карті свого власного зображення для обробки, тобто робота в два потоки. Перший варіант виявився недоцільним для реалізації в зв’язку з необхідністю постійного обміну даними між відеокартами, яка виникає в силу особливостей алгоритму прямого ПІ перетворення. Робота в два потоки, в свою чергу, значно більше відповідає практичним потребам, оскільки за допомогою ПІ перетворення обробляється досить великий набір зображень (як правило, тисячі). Окрім того, при такій організації обчислень майже відсутня потреба в обміні даними між відеокартами, що позитивно відзначається на швидкодії.

Проте використання бібліотеки CUDA Thrust, яка реалізує ПІ оброблення зображень на основі одного GPU, виявилось проблемним через технічні обмеження бібліотеки. Проблема полягає в тому, що майже всі функції цієї бібліотеки є синхронними, тобто блокують всі наступні операції до закінчення своєї роботи. Таким чином нівелюються майже всі переваги Multi-GPU Programming, оскільки ця технологія розрахована в першу чергу на роботу з асинхронними функціями. Таким чином, при використанні синхронних функцій відеокарти починають працювати не паралельно, а по черзі, що призводить до фактичного перетворення паралельної програми у послідовну.

Рішенням цієї проблеми стало комбіноване застосування технології Multi-GPU Programming та відкритого стандарту багатопоточності OpenMP [13,14]. OpenMP є інтерфейсом прикладного програмування для створення багатопоточних додатків, призначених в основному для паралельних обчислювальних систем із спільною пам'яттю та складається з набору директив для компіляторів і бібліотек спеціальних функцій. OpenMP дозволяє легко і швидко створювати багатопотокові програми на алгоритмічних мовах Fortran і C / C ++, при чому директиви OpenMP аналогічні директивам препроцесора для мови C / C ++ і є аналогом коментарів у алгоритмічній мові Fortran.

З практичної точки зору OpenMP дозволяє створювати багатопоточні додатки на рівні центрального процесору (CPU). Робота багатопоточної програми починається з ініціалізації та виконання головного потоку (процесу), який у міру необхідності створює і виконує паралельні потоки, передаючи їм необхідні дані. Паралельні потоки з однієї паралельної області програми можуть виконуватися як незалежно один від одного, так і з пересилкою та отриманням повідомлень від інших паралельних потоків. Остання обставина ускладнює розробку програми, оскільки в цьому випадку програмісту доводиться займатися плануванням, організацією і синхронізацією посилки повідомлень між паралельними потоками. Як пра-

вило, є доцільним виділяти так звані області розпаралелювання, в яких можна організувати виконання незалежних паралельних потоків (рис. 2) [13].

Для обміну даними між паралельними процесами (потоками) в OpenMP використовуються загальні змінні. При зверненні до загальних змінним в різних паралельних потоках можливе виникнення конфліктних ситуацій при доступі до даних. Для запобігання конфліктів можливо використати процедуру синхронізації (synchronization).

Виконання паралельних потоків в паралельній області програми починається з їх ініціалізації. Вона полягає у створенні дескрипторів породжуваних потоків і копіюванні всіх даних з області даних головного потоку в області даних створюваних паралельних потоків. Після завершення виконання паралельних потоків управління програмою знову передається головному потоку [13].

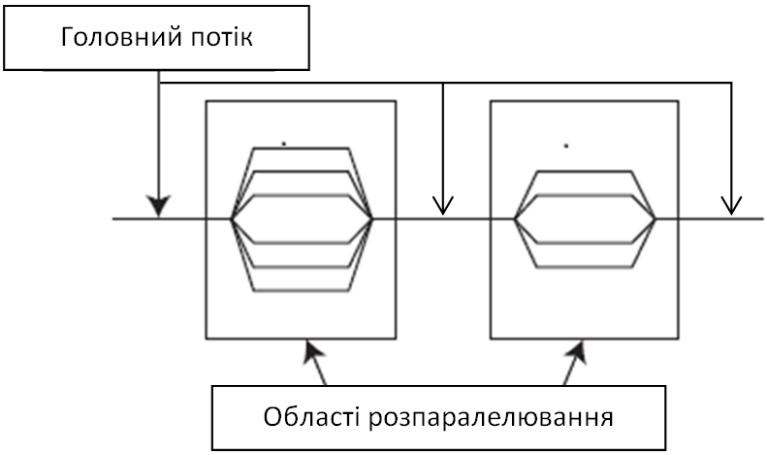

Рисунок 2 – Схема виконання незалежних паралельних потоків

Проте в рамках даного дослідження головною перевагою багатопоточних програм на основі OpenMP виявилась сумісність з NVIDIA CUDA, тобто можливість інтеграції в них CUDA-коду, що дозволило обійти обмеження бібліотеки CUDA Thrust. В результаті, алгоритм роботи комбінованої програми прийняв такий вигляд: спочатку в головному потоці здійснюється попередня обробка зображень та їх розподілення на дві групи, потім за допомогою засобів OpenMP на CPU викликаються два паралельних потоки, кожному з яких за допомогою засобів Multi-GPU Programming ставиться у відповідність своя графічна карта та свій набір зображень. В кожному з цих потоків здійснюється ПІ оброблення відповідної групи зображень за допомогою обчислювальних потужностей відповідної графічної карти за тим самим принципом, що й в програмі на основі одного GPU. Таким чином, синхронність функцій бібліотеки CUDA Thrust перестає відігравати суттєву роль, оскільки кожен потік і кожна відеокарта працюють незалежно один від одного. По закінченню роботи обох потоків отримані ними результати об'єднуються у головному потоці і виводяться користувачеві. Варто відзначити, що комбіноване використання OpenMP та NVIDIA CUDA для задач обробки зображень містить в собі елемент наукової новизни та є перспективною темою для подальших досліджень.

Розроблений програмний модуль підтримує роботу у двох режимах: з використанням одного та двох GPU [15]. Для тестування роботи програми було обрано набір з 20 плямоподібних зображень формату "bmp" з розмірністю 128×128 пікселів. При роботі в першому режимі усі 20 зображень оброблялися на одній графічній карті, при роботі в другому кожна з двох графічних карт отримувала по 10 зображень. З метою зменшення похибки було проведено 10 ітерацій експериментів. За результатами роботи програми середній час виконання в першому режимі становив 452 секунд, а в другому – 238 секунд (рис. 3), що відповідає збільшенню швидкодії в 1,9 разів при використанні двох GPU порівняно з одним GPU.

Висновки

Проведені експерименти показують доцільність реалізації прямого паралельно-ієрархічного перетворення на основі Multi-GPU систем. На основі проведених експериментальних досліджень на базі двох графічних карт було отримано середній приріст швидкодії в 1,9 разів в порівнянні з використанням лише однієї графічної карти. Такий приріст досягається за рахунок нового підходу до організації обчислень, а саме комбінованого використання технологій NVIDIA CUDA та OpenMP. Таке рішення дозволило використати всі переваги бібліотеки CUDA Thrust, незважаючи на синхронність більшості її функцій. Також, варто відзначити перспективність подальших досліджень в напрямі комплексного застосування різних програмних та апаратних засобів досягнення паралелізму. Окрім того, обраний підхід є добре масштабованим, тобто дозволяє реалізувати необхідні обчислення на основі трьох, чотирьох або більше графічних

карт. Результати даного дослідження в подальшому можуть бути використані при створенні комплексних програмно-апаратних засобів аналізу та розпізнавання зображень.

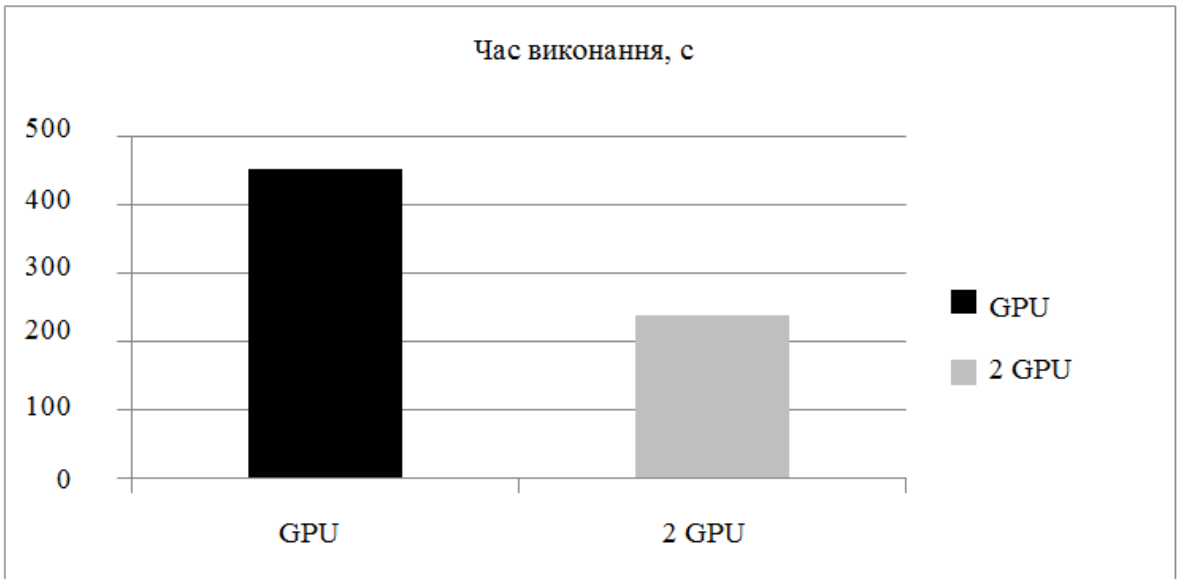

Рисунок 3 – Результати роботи програмного модуля ПІ оброблення зображень з використанням одної та двох графічних карт

Список літератури

1. Паралельно-ієрархічне перетворення як системна модель оптико-електронних засобів штучного інтелекту : [Монографія.] / В.П. Кожем’яко, Ю.Ф. Кутаєв, С.В. Свєчніков, Л.І. Тимченко, А.А. Яровий – Вінниця: УНІВЕРСУМ-Вінниця, 2003. – 324 с.

2. Яровий А.А. Паралельно-ієрархічне перетворення інформаційних середовищ на основі гетерогенної кластерної системи / А.А. Яровий // Вісник ВПІ. – 2011. – № 2 (95). – С. 120 – 127.

3. Кожем’яко В.П. Паралельно-ієрархічні мережі як структурно-функціональний базис для побудови спеціалізованих моделей образного комп’ютера : [Монографія] / В.П. Кожем’яко, Л.І. Тимченко, А.А. Яровий. – Вінниця: Універсум-Вінниця, 2005. – 161 с.

4. M. Galloy CPU vs GPU performance [Електронний ресурс] / M. Galloy // Режим доступу: http://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html.

5. NVIDIA CUDA — Неграфические вычисления на графических процессорах [Електронний ресурс] – Режим доступу: http://www.ixbt.com/video3/cuda-1.shtml.

6. NVIDIA – World Leaderin Visual Computing Technologies [Електронний ресурс] – Режим доступу: http://www.nvidia.ru/page/home.html.

7. OpenCL – The open standard for parallel programming of heterogeneous systems [Електронний ресурс] – Режим доступу: https://www.khronos.org/opencl/.

8. Яровий А. А. Паралельно-ієрархічні мережі на основі формування нормуючого рівняння з контрольованим навчанням для класифікації зображень профілю лазерного променя / Яровий А. А., Тимченко Л.І., Матвійчук М.С. // Інформаційні технології та комп’ютерна інженерія. – 2014. – №3(31). – С. 78-86.

9. Яровий А.А. Паралельно-ієрархічне перетворення плямоподібних зображень на основі GPU-орієнтованої апаратної платформи / Яровий А.А., Арсенюк І.Р., Матейчук М.С., Кашубін С.Г., ПольгульТ. Д. // Вісник Хмельницького національного університету. Технічні науки. – 2014. – №6(219). – С. 127-133.

10. A. Yarovyy, L. Timchenko, N. Kokriatskaia, S. Nakonechna, M. Mateichuk Organization of High-Performance Parallel-Hierarchical Computing Processes for Classification of Laser Beam Images. – Development and application systems : Proceedings of the 12th International Conference on DAS-2014, May 15-17, 2014, Suceava, Romania – Suceava, Universitatea Stefan cel Mare Suceava, 2014 – p. 192-197.

11. Перспективи застосування технології NVIDIA SLI для паралельно-ієрархічної обробки зображень / Яровий А.А., Кулик О.О. : Збірник тез доповідей VII Міжнародної науково-технічної конференції [Фотоніка ОДС-2015)], (Вінниця, 21-23 квітня 2015 р.) – Вінниця, ВНТУ, 2015. – с. 10.

12. Thrust – Parallel Algorithms Library [Електронний ресурс] – Режим доступу: https://thrust.github.io/.

13. Левін М. А. Параллельное программирование с использованием OpenMP [Електронний ресурс]. – Режим доступу: http://www.intuit.ru/studies/courses/1112/232/info.

14. The OpenMP® API specification for parallel programming [Електронний ресурс] – Режим доступу: http://openmp.org/wp/.

15. Свідоцтво про реєстрацію авторського права на твір № 62201. Комп’ютерна програма "Комп’ютерна програма обробки зображень у GPU-орієнтованому апаратному забезпеченні на основі модифікованого для Multi-GPU Programming методу прямого паралельно-ієрархічного перетворення" / Яровий А.А., Кулик О.О., Матейчук М.С. Дата реєстрації Державною службою інтелектуальної власності України 23.10.2015.

Відомості про авторів

Яровий Андрій Анатолійович – д.т.н., професор, професор кафедри комп’ютерних наук, Вінницький національний технічний університет, м. Вінниця, Хмельницьке шосе, 95.

Кулик Олександр Олександрович – магістрант кафедри комп’ютерних наук, Вінницький національний технічний університет, м. Вінниця, Хмельницьке шосе.

Кокряцька Наталія Іванівна – к.т.н., доцент, доцент кафедри телекомунікаційних технологій та автоматики, державний економіко-технологічний університет транспорту, м. Київ.