УДК 004.4

РЕАЛІЗАЦІЯ ГІБРИДНОЇ АРХІТЕКТУРИ КЛАСТЕРНИХ ОБЧИСЛЕНЬ

В. Парубочий, Р. Шувар

Кафедра системного проектування,
Львівський національний університет імені Івана Франка
вул. Драгоманова, 50, 79005 Львів, Україна
eddragonwolf@ukr.net

У статті розглядаються принципи організації гібридної архітектури кластерних обчислень на основі використання обчислювальних можливостей центрального процесора і графічного процесора, з підтримкою технології GPGPU. Проаналізовано переваги та недоліки такого методу обчислень, а також показано можливості реалізації паралельних програм на основі використання гібридної архітектури кластерних обчислень. На основі запропонованої програмної моделі реалізовано програмний каркас, який може бути використаний для розробки прикладних програм. З метою демонстрації можливості практичного використання гібридної архітектури, а також для аналізу її ефективності, на основі запропонованого програмного каркасу здійснено реалізацію програми блокового Фур'є-перетворення для зображення високої роздільної здатності. Результати опрацювання, а також обґрунтування і порівняння результатів наведено в останній частині статті.

Ключові слова: гібридна архітектура, кластер, CUDA, CTM, OpenCL, MPI, NVIDIA, GPGPU, FFTW, CUFFT.

Паралельні обчислення — широка область програмування, яка дає змогу вирішити ряд задач обробки і аналізу великих об'ємів даних, розв'язок яких методами послідовного програмування утруднюється через великі обсяги пам'яті і часу, що необхідні для розв'язання поставлених задач. Паралельні обчислення дають змогу не лише обійти ці обмеження за допомогою розділення задачі на ряд підзадач меншої розмірності, а й досягнути прискорення обчислень за рахунок паралельного виконання підзадач і оптимізації обміну даних.

Серед усіх областей паралельного програмування, на сьогоднішній день, кластерні обчислення є найбільш перспективним і доступним напрямком розвитку ідей паралельного програмування.

Практичне застосування і широка область задач, яка з кожним роком тільки розширюється, зумовила зростання зацікавленості світових виробників відеоприскорювачів до можливостей паралельних обчислень і, як наслідок, появи

технології неграфічних розрахунків загального призначення на графічних процесорах (GPGPU, General-Purpose computation on GPUs). Можливість розпаралелювання обчислень на графічних процесорах досягається завдяки структурі ядра мультипроцесора GPU, що працює за принципом SIMD (single instruction, multiple data; один потік команд, багато потоків даних) і є, по суті, паралельним процесором, що дає змогу виконувати спеціальні паралельні алгоритми обробки даних.

Саме тому, уже доволі давно (приблизно з 80-х років XX століття) існує значний інтерес наукової спільноти та розробників програмних продуктів до можливостей реалізації математичних обчислень на графічних процесорах. Проте відсутність доступних і відносно простих рішень у даній області до недавнього часу значно сповільнювала рух інформаційних технологій у цьому напрямку. Однак вихід декількох оптимальних рішень технології GPGPU спричинив стрімке зростання зацікавленості до можливостей цих апаратних архітектур і технологій. Однією з таких технологій реалізації паралельних неграфічних обчислень на графічних процесорах є технологія CUDA (Compute Unified Device Architecture) від компанії NVIDIA, яка викликала значне зацікавлення у світової спільноти. Свідченням цьому є велика частина статей присвячених даній темі. Зокрема, можна відмітити результати робіт [1-5], в яких розглядається широкий спектр питань пов'язаних з реалізацією програм за допомогою технології CUDA, а також створення кластерів на базі CPU і GPU.

Поява рішень відеокарт з можливістю неграфічних обчислень відкрила шлях для розвитку нових методів паралельних обчислень не лише на персональних комп'ютерах, оскільки оснащений такою відеокартою ПК навіть без наявності багатоядерного процесора можна використовувати як паралельну обчислювальну машину малої потужності, а й до виникнення нових архітектурних рішень і методів організації кластерних обчислень, прикладами яких є такі обчислювальні системи, як Titan (Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x) і Piz Daint (Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x) [6].

Крім того, графічні процесори розвиваються набагато швидше, ніж центральні. Так, у квітні 2016 року NVIDIA анонсувала п'яте покоління архітектури CUDA – Pascal, а уже через два роки очікується поява наступного покоління — Volta. У першу чергу, зростають обчислювальні можливості — збільшується кількість обчислювальних ядер, особливо з подвійною точністю (так процесор GP100 на архітектурі Pascal має практично у два рази більше ядер з подвійною точністю, ніж, наприклад, один з його найбільш вдалих попередників — процесор GK110 на архітектурі Kepler (3 покоління)), збільшується розмір кешу другого рівня і розрядність інтерфесу пам'яті, що, у свою чергу, веде зростання швидкодії обміну даних. Ще однією особливісю нової архітектури є перехід до 16-нм технологічного процесу, що дав змогу подвоїти кількість транзисторів на чіпі (GK110 (Kepler) – 7.1 млд., GM200 (Maxwell) – 8 млд., GP100 – 15.3 млд.).

З технічної точки зору, можна виділити декілька параметрів графічних процесорів, які значною мірою впливають на їхні обчислювальні здатності:

- кількість обчислювальни ядер з одинарною і подвійною точністю (FP32 і FP64 Cores);
- пікова кількість операцій з плавачою комою (одинарною і подвійною точністю, Peak FP32 GFLOPs і Peak FP64 GFLOPs);
- інтерфейс пам'яті (Memory Interface) — розрядність і тип пам'яті (GDDR або HBM);
- розмір пам'яті (Memory Size);

- розмір кешу другого рівня (L2 Cache Size).

З програмної точки зору можна класифікувати графічні процесори з залежності від програмних інтерфейсів (CUDA, OpenCL, OpenGL, DirectX) і їхньої максимальної версії, яку підтримує процесор.

Окрім цих характеристик, архітектура CUDA надає ряд можливостей, що можуть значно оптимізувати параметри архітектури. Зокрема, архітектура Pascal підтримує уніфіковану і 3D пам'ять, Уніфікована пам'ять дає змогу працювати в межах одного адресного простору як на графічному процесорі, так і центральному, а також може розширити обсяг пам'яті графічного прискорювача за рахунок загальної пам'яті. 3D-пам'ять це особливий спосіб організації пам'яті, що дає змогу оптимізувати побудову нейронних мереж і алгоритмів машинного навчання на графічних процесорах [15].

Одним з нових методів організації обчислень є гібридна архітектура кластерних обчислень. Основна ідея гібридної архітектури організації обчислень базується на твердженні, що оскільки значна частина задач великої розмірності, що розв'язуються методами паралельного програмування, завдяки своїй структурі, може бути розв'язана на графічних процесорах зі значно нижчими затратами часу, ніж на центральному процесорі, то можна досягти значного збільшення швидкодії і рівня паралелізму кластерних обчислень за рахунок перенесення обчислення графічно-адаптивних задач чи їхніх частин на графічні процесори з підтримкою технології GPGPU.

Це твердження дає змогу виділити наступні рівні розпаралелювання у гібридній архітектурі (Рис. 1):

Розпаралелювання на рівні кластера полягає у розбитті вихідної задачі на підзадачі, що можуть виконуватися паралельно, і передачі цих підзадач вузлам кластера. На противагу від класичної реалізації кластерних обчислень, де розпаралелювання на рівні кластера виконує головну роль у розбитті задачі на паралельно виконувані підзадачі, у гібридній архітектурі він виконує лише узагальнене розбиття задачі на основі виділення головних паралельно виконуваних частин задачі. Основне розпаралелювання відбувається на нижчих рівнях, оскільки саме на них повністю враховуються особливості паралельних алгоритмів, адаптованих під виконання на графічних процесорах.

Розпаралелювання на рівні вузла складається з двох етапів: перший полягає у аналізі того чи можна виконати отриману задачу за допомогою обчислювальних можливостей GPU, а другий етап включає у себе пошук усіх доступних для даного вузла графічних процесорів з підтримкою здійснення математичних обчислень, розбитті поточної задачі на підзадачі, які графічні процесори можуть виконувати паралельно, і передачі цих підзадач на графічні процесори.

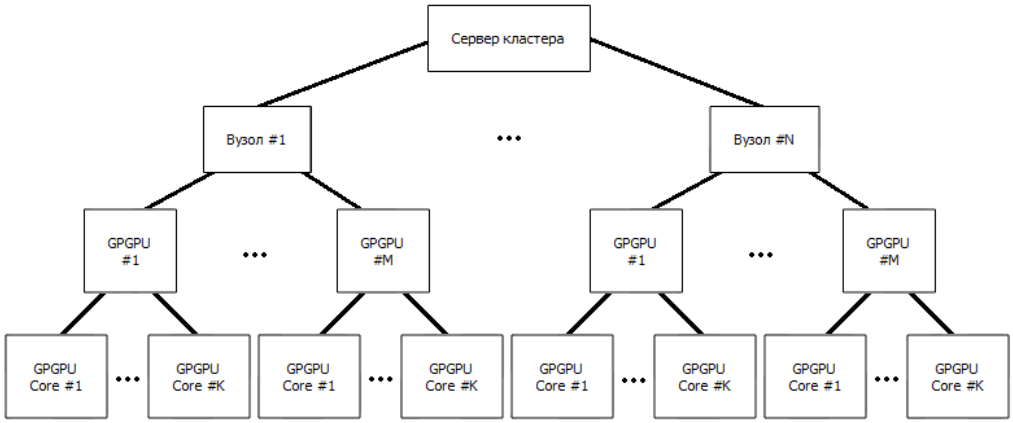

Рис. 1. Гібридна архітектура кластерних обчислень.

Важливою рисою виконання розпаралелювання задач на даному рівні є можливість розподілення підзадач не лише між графічними процесорами, а й між центральним процесором чи процесорами. Тобто, на даному етапі задачу можна розбити спочатку на дві логічні частини: частину, що буде виконуватися графічним процесором (процесорами), і частину, що виконуватиметься паралельно чи послідовно на центральному процесорі, а потім здійснити додаткове розпаралелювання за рахунок використання декількох графічних і (чи) центральних процесорів.

Розпаралелювання на рівні GPU здійснюється створенням спеціальних паралельних алгоритмів адаптованих під конкретну архітектуру графічних процесорів.

На перший погляд, така кількість рівнів може призвести до зростання часу виконання програми за рахунок збільшення часу передачі даних між рівнями і обміну даних на одному рівні. Проте, по-перше, більшість практичних задач, що розв'язуються на кластерах сьогодні, можна доволі просто і ефективно розбити на незалежні підзадачі з мінімальним рівнем взаємодії (він включатиме лише отримання даних для обробки і повернення результатів після неї). По-друге, на відміну від обміну даних між вузлами кластера, обмін даних між мікропроцесорами на рівні GPU відбувається значно швидше за рахунок декількох рівнів організації пам'яті, яка в середньому має значно вищу пропускну здатність у порівнянні з оперативною пам'яттю на вузлах кластера. По-третє, сучасні рішення відеокарт як з підтримкою неграфічних обчислень, так і без них, мають значний об'єм пам'яті, яка може бути використана у процесі обчислення, що дає змогу не лише вирішувати задачі, структура яких не дає змогу здійснити розбиття на велику кількість малих підзадач, а дає змогу виділити лише декілька підзадач, але й задачі, розв'язок яких потребує складних методів обробки і аналізу, що в свою чергу можуть вимагати доволі великих об'ємів пам'яті для збереження проміжних чи тимчасових даних.

Оскільки реалізація паралельних обчислень на кожному рівні архітектури залежить безпосередньо від апаратної частини, на якій будуть реалізовані паралельні алгоритми, кожен рівень гібридної архітектури включає свої апаратні і програмні засоби. Крім того, більшість програмних засобів, що використовуються для паралельного програмування, є низькорівневими мовами програмування чи надбудовами над звичайними мовами програмування з високим степенем інтеграції у апаратну базу, на якій вони реалізовуються.

Так на рівні кластера апаратна частина складається з вузлів кластера, що є окремими комп'ютерами, пов'язаними між собою певним видом мережі. Програмна частина реалізується за допомогою мови програмування C (C++) і технології передачі повідомлень MPI (Message passing interface), що здійснює обмін даних між вузлами кластера. До рівня кластера можна також віднести і сервер кластера (див. Рис. 1). Тут під сервером мається на увазі головний вузол кластера (зазвичай з індексом 0), який здійснює операції розділення вихідної задачі на ряд підзадач, надсилання підзадач іншим вузлам, прийому від цих вузлів результатів їхньої обробки і формування кінцевого результату вихідної задачі.

Апаратна частина на рівні вузла складається з ПК, який містить одну чи декілька відеокарт з підтримкою неграфічних обчислень. Програмна реалізація включає в себе мову програмування C (C++) і програмні засоби конкретної реалізації технології GPGPU на наявних відеокартах. Хоча на даному рівні архітектури вже відбувається зв'язок з графічними процесорами, він полягає у представленні наявної на даному вузлі задачі у вигляді, в якому вона може бути виконана на графічному процесорі, і передачі її на виконання.

Апаратна і програмна реалізація останнього рівня залежить, як зазначалось вище, безпосередньо від конкретної реалізації технології GPGPU на наявних відеокартах. Сьогодні найбільш прогресивними реалізаціями технологіями GPGPU є технологія CUDA (Compute Unified Device Architecture) [7], що була анонсована компанією NVIDIA в 2006 році і підтримується на ряді відеокарт NVIDIA (серії відеокарт GeForce 8, GeForce 9, GeForce 200, NVS, Quadro і Tesla [8]) і технологія ATI Stream Technology (або ATI FireStream і AMD Stream Processor), розроблена компанією AMD [9].

Технологія ATI Stream Technology молодша за технологію CUDA і прийшла на зміну іншій технології від AMD - CTM (Close To Metal або AMD Stream Computing) [10]. Хоча технологія CTM випередила своєю появою навіть технологію CUDA, проте вона мала ряд недоліків, основним з яких був підхід до програмування. Тоді як CUDA - це розширення мови програмування C, CTM є віртуальною машиною, що виконує асемблерний код. Технологія ATI Stream Technology була створена з метою виправити цей та інші недоліки CTM, тому її програмним інтерфейсом стає програмний каркас OpenCL [11]. OpenCL – це мова програмування для розробки паралельних програм на центральних і графічних процесорах, створена на основі стандарту C99 консорціумом Khronos Group. Основною перевагою OpenCL є гібридність, оскільки один і той же код може бути використаний як на центральному процесорі, так і на графічному. Окрім цього, ATI Stream Technology перспективно має значно кращий рівень швидкодії на графічних процесорах від AMD. Для прикладу, AMD Radeon HD6990 дає 5,1 Tflops для операцій з одиничною точністю і 1,2 Tflops для операцій з подвійною [12], тоді як одне з найпотужніших рішень від NVIDIA - Tesla K40 лише 4,29 Tflops і 1,43 Tflops відповідно [13].

Тим не менше, CUDA — це технологія, яка не лише завоювала свою долю ринку і значне число прихильників. Це технологія, яка поряд зі зручністю і відносною простотою програмування, включає спеціалізоване програмне забезпечення, зокрема Just-in-time асемблер PTX-коду, середовище розробки у вигляді компілятора nvcc, бібліотеку підтримки виконання (runtime), cudart, профілювач NSight, відлагоджувач gdb для GPU та прикладні, зокрема математичні, бібліотеки: cuFFT, cuBLAS, cuSPARSE, cuSOLVER, cuRAND, nvGRAPH, Thrust, NPP, cuMath, реалізації LAPACK, а також велике число літератури і додаткової інформації від NVIDIA і сторонніх розробників, що значно полегшує процес створення і відлагодження CUDA-програм.

Ці передумови, а також велика популярність рішень відеокарт від компанії NVIDIA, зумовили наш вибір технології CUDA для реалізації гібридної архітектури кластерних обчислень і дослідження можливості більш ефективного вирішення поставлених задач за допомогою графічного процесора.

Загальний опис архітектури, навіть без деталізації процесу розробки паралельних алгоритмів, дає змогу виділити ряд важливих плюсів і мінусів гібридної архітектури кластерних обчислень.

Перевагами гібридної архітектури у порівнянні з класичними методами організації кластерних обчислень є:

Підвищення рівня паралелізму, що досягається за рахунок введення додаткового рівня розпаралелювання поставленої задачі. Класична організація обчислень на кластері включає лише один-два рівні розпаралелювання, перший з яких полягає у розбитті задачі на підзадачі і передачі їх на виконання на інші вузли кластера, а другий рівень можливий тоді, коли центральний процесор на кожному вузлі має можливість паралельного виконання задач. Гібридна архітектура дає змогу виділити третій рівень розпаралелювання вихідної задачі (чи її підзадач) за рахунок обробки виділеної підзадачі на графічному процесорі. При цьому, в загальному випадку, можливе паралельне виконання не лише задач, що розв'язуються на графічних процесорах, а й паралельне виконання підзадач, що виконуються на центральному і графічному процесорах.

Зменшення формальних обмежень щодо розробки паралельних алгоритмів пов'язаних з апаратними і програмними засобами кластерів. На перших етапах створення паралельних алгоритмів розробники зустрічались з цілим рядом проблем пов'язаних з організацією незалежного виконання підзадач на різних вузлах кластера і одночасно з забезпеченням ефективної взаємодії та обміну даними між вузлами. Ще однієї проблемою було ефективне використання машинного часу і синхронізації роботи вузлів кластера, оскільки ефективність використання паралельних обчислень істотно знижується, якщо якийсь з вузлів повинен очікувати на дані з іншого вузла, або якщо вузли закінчують обробку своїх даних швидше і очікують доки свої завдання закінчать усі вузли для формування кінцевого результату. І хоча більшість з цих проблем були вирішенні завдяки розробці спеціальних принципів організації програм і їхніх підпрограм, що повинні були виконуватися паралельно, а обмін даними між вузлами був оптимізований і стандартизований завдяки технології передачі повідомлень MPI, розробникам паралельних програм так і не вдалося подолати однієї з найбільших проблем — поганого розпаралелювання ряду практичних задач. Попри великий клас паралельно-роздільних задач, ще більша частина задач не піддається простому розпаралелюванню і потребує інших підходів, які не вкладаються у класичні методи організації паралельних обчислень.

Введення рівня паралельного виконання підзадач за допомогою можливостей графічного процесора, хоч і не вирішує цю проблему у повній мірі, проте дає змогу ефективно обробляти дані, структура яких близька до моделі обробки інформації на графічних процесорах. До таких даних можна, в першу чергу, віднести різноманітні графічні дані, які потребують неграфічної обробки. До класу задач, які можна ефективно виконувати на графічних процесорах можна віднести також і паралельні задачі з високим рівнем взаємодії між підзадачами і коротким часом виконання самих підзадач. Цей клас задач дає змогу виділити ще одну перевагу гібридної архітектури, а саме ***оптимізацію обміну даних***. Хоча такі задачі і піддаються до розпаралелювання, проте через високий рівень обміну даних їхня оптимальність знижується, оскільки навіть за

наявності каналів високої пропускної здатності і використання технології передачі повідомлень MPI, операції прийому-передачі даних є часозатратними і потребують додаткової синхронізації вузлів кластера. Гібридна архітектура дає змогу уникнути складного обміну даних на рівні вузлів кластера за рахунок перенесення його на більш нижчий рівень — рівень паралельного виконання на графічних процесорах, де обмін даних може здійснюватися набагато швидше, завдяки спільній пам'яті, до якої мають доступ усі ядра графічного мультипроцесора.

Теоретичне ***підвищення швидкодії виконання алгоритмів*** також можна віднести до переваг гібридної архітектури. Хоч дане твердження і володіє деякою неоднозначністю, пов'язаною, насамперед, з тим, що збільшення швидкодії за рахунок використання графічного процесора досягається лише для тих задач, які просто адаптуються для виконання на графічних процесорах, не можна заперечувати, що показники швидкодії і збільшення паралелізму у графічних процесорів набагато вищі, ніж в універсальних процесорів. Саме це в поєднанні з класом графічно-адаптивних задач, який зі збільшенням потреб у обробці великих об'ємів даних кожного року збільшується, дає змогу оптимістично оцінювати можливе збільшення швидкодії паралельного виконання алгоритмів на графічних процесорах і гібридній архітектурі кластерних обчислень зокрема.

Окрім обчислювальних переваг гібридної архітектури, можна виділити і економічний чинник, що полягає у ***доступності відеокарт з можливістю неграфічних розрахунків загального призначення*** на графічних процесорах. Світові лідери по виробництву відеокарт NVIDIA і AMD активно впроваджують свої варіанти технології GPGPU у практичне використання, за рахунок забезпечення підтримки цих технологій усіма новими рішеннями відеокарт. Це зумовлює те, що завдяки популярності і поширеності продукції цих компаній, можна отримати потужну паралельну обчислювальну машину за значно нижчу ціну, ніж у випадку реалізації кластера виключно на центральних процесорах.

Ще однією позитивною рисою, яку привносять паралельні обчислення на графічних процесорах, є поступовий вихід паралельного програмування на певний стандартизований рівень, що базується на мовах програмування C (C++) і ряді надбудов над цими мовами програмування, що не лише спрощує взаємозв'язок між різними частинами програми, а й дає змогу сподіватися на появу програмних засобів, які міститимуть у собі усі необхідні модулі та бібліотеки і забезпечуватимуть спрощення розробки паралельних алгоритмів.

Попри усі переваги, внесення у архітектуру додаткового рівня розпаралелювання загострює ряд проблем організації кластерних обчислень, які у поєднанні з недоліками і обмеженнями технології GPGPU у тому чи іншому виконанні, дають змогу виділити ряд недоліків гібридної архітектури організації кластерних обчислень.

Так додатковий рівень розпаралелювання і програмні засоби різних рішень технології GPGPU ускладнюють і так непросту структуру паралельної програми. Це проявляється, насамперед, у тому, що однорідна структура програми (у випадку класичних методів організації кластерних обчислень) розділяється на декілька частин (у гібридній архітектурі кластерних обчислень). Основні виконувані частини такої програми — це частина розпаралелювання на рівні вузлів кластера, де виконується виділення підзадач і обміну даними за допомогою технології MPI, і частина розпаралелювання на рівні GPU, яка реалізує алгоритм розв'язання підзадачі на конкретній реалізації технології GPGPU. Між цими частинами виділяється ще один шар — адаптивний — який не здійснює обробки даних, а спочатку перетворює задачу у

вигляд зручний для виконання на графічному процесорі, а потім видозмінює результати обробки на рівні GPU таким чином, щоб вони могли бути використані для отримання кінцевого результату на рівні вузлів кластера.

Ускладнення структури програми зумовлює підвищення складності розробки самого паралельного алгоритму. Це зумовлено тим, що паралельні алгоритми чи точніше засоби, якими вони реалізуються, сильно пов'язанні з апаратною частиною обчислювальної машини. У цьому плані рішення технології GPGPU не є виключеннями. Хоч вони і спрощують ряд особливостей роботи з графічними процесорами, повноцінне використання обчислювальних потужностей GPU в них можливе лише за рахунок низькорівневого доступу до пам'яті і використання спеціальних команд графічного процесора. Тому, очевидно, що практичне використання гібридної архітектури ускладнюється потребою підготовки професійних кадрів, які б могли врахувати не лише особливості паралельного програмування на вузлах кластера, а й специфіку програмування на графічному процесорі.

Низька абстрагованість програмних засобів розробки алгоритмів на GPU від апаратної частини зумовлює ще один недолік гібридної архітектури, а саме низьку переносимість програми. Зокрема, згадані запропоновані технології від компаній NVIDIA і AMD є несумісними між собою, що до недавнього часу унеможливлювало використання програми написаної на CUDA на графічних адаптерах компанії AMD і навпаки. Тим не менше, хоч кінцевого рішення ще не має, але NVIDIA і AMD уже почали роботу для вирішення цієї проблеми. Так останні версії набору інструментів від NVIDIA уже підтримують компіляцію і виконання коду на програмному інтерфейсі ATI OpenCL, хоч і з втратою швидкодії. У свою чергу, AMD надає значний обсяг інформації про транслювання CUDA-коду в програмний інтерфейс ATI OpenCL [14]. Проте, несумісність рішень відеокарт на апаратному рівні від цих виробників, накладає обмеження на саму структуру кластера, вимагаючи використання у всьому кластері графічних процесорів, виконаних лише за однією технологією. Цей факт є слабким місцем гібридної архітектури, оскільки класичні методи організації кластерних обчислень поступово долають проблему переносимості за рахунок використання кросплатформних засобів і уніфікації стандартів обміну та організації даних.

Ці недоліки показують деяку слабкість гібридної архітектури у плані широкого практичного застосування, проте зовсім не заперечують ефективності використання графічним процесорів з можливістю математичних обчислень для реалізації кластерних обчислень. Більше того, під час огляду проблеми ускладнення структури паралельної програми подається напрямок для можливого вирішення цих недоліків гібридної архітектури. Мається на увазі розділення програми на три частини (Рис. 2): частину, що виконується на рівні вузлів кластера (кластерний шар); частину, що виконується на рівні GPU (шар GPU); середню частину, що здійснює перехід між першими двома частинами (адаптивний шар).

Кожна з цих частин базується на окремому рівні гібридної архітектури, а отже використовує лише обмежену частину програмних засобів, які необхідні для розробки програми.

Так кластерний шар, що використовує для реалізації розпаралелювання початкової задачі на ряд підзадач, надсилання цих підзадач усім вузлам кластера, обмін даними під час виконання обробки і нарешті отримання результатів роботи від усіх вузлів для формування кінцевого результату, використовує всього два засоби: мову програмування C (C++) і бібліотеку реалізації технології передачі повідомлень MPI.

Шар GPU використовує лише один засіб — засіб, який надається розробником відеокарти, і призначений для розробки програм на даній реалізації технології GPGPU.

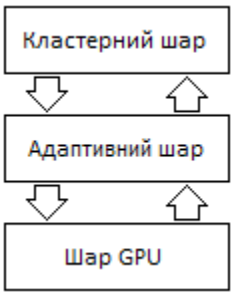

Рис. 2. Модель паралельної програми, реалізованої на гібридній архітектурі кластерних обчислень.

Адаптивний шар використовує основні програмні засоби з двох інших шарів для представлення даних, оскільки він здійснює перетворення представлення даних кластерного шару у ті типи даних, що використовуються на шарі GPU, і зворотну операцію над результатами обробки на шарі GPU для подальшого аналізу і обробки на кластерному шарі.

З метою дослідження можливості використання графічних процесорів з технологією CUDA на базі кластера паралельних та розподілених обчислень Львівського національного університету імені Івана Франка для реалізації кластерних обчислень було практично реалізовано гібридну архітектуру паралельних обчислень. Розроблено програмний каркас для спрощення процесу реалізації паралельних алгоритмів, орієнтованих на виконання на кластерах з гібридною архітектурою. Програмний каркас демонструє реалізацію моделі паралельної програми, реалізованої на гібридній архітектурі кластерних обчислень, і може бути використаний в якості основи для розробки більш складних програм. Також здійснено порівняльну характеристику реалізації швидкого дискретного Фур'є-перетворення на гібридній і класичній архітектурі організації кластерних обчислень.

Для практичної реалізації гібридної архітектури використано наступну конфігурацію кластера:

Сервер кластера - чотирьохядерний процесор Intel Xeon X3440 із частотою 2,53 ГГц, 16 ГБ ОЗП DDR3; 14 вузлів на базі двоядерного AMD Athlon II із частотою 2,9 ГГц, 8 ГБ ОЗП DDR3, GP GPU nVidia GeForce GTS, 1024 МБ ОЗП GDDR5, об'єднаних локальною комп’ютерною мережею Gigabit Ethernet cat. 5e для керування кластером і локальною комп’ютерною мережею Gigabit Ethernet cat. 6 для обміну повідомленнями між вузлами кластера.

Програмне забезпечення включає ОС Scientific Linux 6.2, ядро версії 3.4, систему черг та розподілу ресурсів SGE v.6.2u5, систему адміністрування кластера Webmin v.1.580, систему моніторингу кластера Ganglia v.3.1.7, пакету компіляторів GCC v.4.4.6, ПЗ для паралельних обчислень OpenMPI v.1.5.3, бібліотеку реалізації швидкого Фур'є-перетворення FFTW v.3.2.1 і пакет CUDA Toolkit для реалізації паралельних обчислень на графічних процесорах. Експерименти проводились на версії CUDA Toolkit 5 Production Release, проте код програми сумісний з усіма наступними версіями набору інструментів CUDA (на разі актуальним є CUDA Toolkit 7.5 і очікується вихід CUDA Toolkit 8).

Реалізація каркасу на рівні кластера полягає в ініціалізації MPI середовища виконання програм і виділення задач для кожного процесора (Лістинг 1).

Апаратна частина реалізації паралелізму на рівні вузла полягає у забезпеченні вузлів кластера графічними процесорами, що здатні реалізовувати математичні обчислення, а програмна реалізація безпосередньо залежить від обраної архітектури графічних процесорів (Лістинг 2).

На основі каркасу програми гібридної архітектури на базі кластера паралельних та розподілених обчислень Львівського національного університету імені Івана Франка було реалізовано програму для виконання двовимірного швидкого дискретного Фур'є-перетворення, яка може виконуватися у двох режимах: з використанням можливостей графічних процесорів від компанії NVIDIA і бібліотеки CUFFT, що включена в програмний пакет CUDA Toolkit, розробленого компанією NVIDIA, а також з використанням бібліотеки FFTW, яка реалізує швидке дискретне перетворення Фур'є на центральному процесорі.

```
char processor_name[MPI_MAX_PROCESSOR_NAME];
MPI_Status stat;
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Get_processor_name(processor_name, &nameLength);
MPI_Comm_rank(MPI_COMM_WORLD,&id);
if (id == 0)
{
//Блок коду процесора з нульовим індексом
}
else
{
//Блок коду усіх інших процесорів
}
MPI_Finalize();
```

Лістинг 1. Блок програми для ініціалізації MPI середовища.

```
cudaGetDeviceCount(&devCount);
if (devCount == 0)
{
//Блок коду при відсутності графічних процесорів, що
підтримують технологію CUDA.
}
else
{
//Отримання властивостей графічного пристрою з номером index
      cudaDeviceProp devProp;
      cudaGetDeviceProperties(&devProp, i);
//Блок коду графічного пристрою з номером index
}
```

Лістинг 2. Блок програми для ініціалізації графічних пристроїв CUDA.

Дослід полягав у покроковому виділенні на зображенні розміром 4688×3248 пікселів квадратних блоків розміром n×n (Значення n наведені в першому стовпчику таблиці 1) і виконання над кожним з цих блоків двовимірного дискретного Фур'є-перетворення. На практиці це було реалізовано таким чином, що процес, який здійснює управління всією програмою, передає іншим процесам координату верхнього лівого кута блоку, а кожен з паралельних процесів, що здійснюють обчислення, формує по координаті лівого верхнього кута і лінійних розмірів відповідний блок і здійснюється над ним Фур'є-перетворення. Після отримання спектру процес надсилає повідомлення про готовність до виконання нових обчислень головному процесу, який надає координату наступного блоку, або надсилає повідомлення про закінчення роботи даного процесу, у випадку якщо досягнуто кінця зображення. Такий метод формування блоків зображення називається методом вікон з одиничним кроком зміщення вікна — кожен наступний блок зміщується відносно попереднього на один піксель по горизонталі, або по вертикалі, а у випадку досягнення кінця зображення метод здійснює перехід на наступний рядок або стовпець.

Оскільки програмний інтерфейс FFTW і CUFFT дуже подібний, реалізація Фур'є-перетворення здійснюється за подібною схемою, з тим виключенням, що для своєї роботи CUFFT потребує копіювання даних з глобальної пам'яті комп'ютера у пам'ять графічного процесора. Для порівняння складності реалізації Фур'є-перетворення за допомогою FFTW і CUFFT у лістингах 3 і 4 відповідно наводиться код реалізації двовимірного Фур'є-перетворення над блоком даних.

```
int size = height * width;
//Об'явлення змінних
fftw_complex *in;
fftw_complex *out;
fftw_plan g;
//Виділення пам'яті
in = (fftw_complex *)fftw_malloc(sizeof(fftw_complex) * size);
out = (fftw_complex *)fftw_malloc(sizeof(fftw_complex) * size);
//Ініціалізація вхідних даних
int k = 0;
for (int i = 0; i < height; ++i)
{
      for (int j = 0; j < width; ++j)
      {
            in[k][0] = block[i][j];
            in[k][1] = 0;
            k++;
      }
}
//Створення плану
g = fftw_plan_dft_2d(height, width, in, out, FFTW_FORWARD,
FFTW_MEASURE);
//Виконання плану перетворення
fftw_execute(g);
```

Лістинг 3. Реалізація двовимірного Фур'є-перетворення за допомогою бібліотеки FFTW.

```
float *hInputBlock;
hInputBlock = (float *)malloc(sizeof(float) * size);
int k = 0;
for (int i = 0; i < height; ++i)
{
      for (int j = 0; j < width; ++j)
      {
            hInputBlock[k] = block[i][j];
            k++;
      }
}
float *dInputBlock;
cudaMalloc((void**)&dInputBlock, sizeof(float) * size);
cudaMemcpy(dInputBlock, hInputBlock, sizeof(float) * size,
cudaMemcpyHostToDevice);
cufftComplex *outputBlock;
cudaMalloc((void**)&outputBlock, sizeof(cufftComplex) * size);
if (cudaGetLastError() != cudaSuccess)
{
      return 0;
}
cufftHandle fftPlan;
if (cufftPlan2d(&fftPlan, width, height, CUFFT_R2C) !=
CUFFT_SUCCESS)
{
      return 0;
}
if (cufftSetCompatibilityMode(fftPlan,
CUFFT_COMPATIBILITY_NATIVE)!= CUFFT_SUCCESS)
{
      return 0;
}
int res = cufftExecR2C(fftPlan, (cufftReal*)dInputBlock,
outputBlock);
if (res != CUFFT_SUCCESS)
{
      return 0;
}
if (cudaThreadSynchronize() != cudaSuccess)
{
      return 0;
}
cufftComplex *hOutBlock;
hOutBlock = (cufftComplex *)malloc(sizeof(cufftComplex) *
size);
cudaMemcpy(hOutBlock, outputBlock, sizeof(cufftComplex) * size,
cudaMemcpyDeviceToHost);
```

Лістинг 4. Реалізація двовимірного Фур'є-перетворення за допомогою бібліотеки CUFFT.

Для запуску програми на виконання використовувалась система черг та розподілу ресурсів SGE. Для її використання було написано shell-скрипт (Лістинг 5). Надсилання програми здійснювалось командою:

qsub -pe orte 14 run.sge

Після запуску цього скрипта програма розміщується у черзі і очікує дозволу на виконання. Результати виводяться у вигляді текстового файлу, який матиме для даного скрипта назву *run.sge.o**, де * - номер, який надала система SGE завданню під час запуску.

```
#!/bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
/usr/lib64/openmpi/bin/mpirun
$HOME/projects/HCCAFFTExperiment/HCCAFFTExperiment
```

Лістинг 5. Shell-скрипт для запуску завдань у системі черг та розподілу ресурсів SGE.

Для дослідження ефективності використання гібридної архітектури для виконання паралельних обчислень було здійснено ряд тестів, в результаті яких приведено в таблиці (таб. 1), яка показує залежність середнього часу виконання швидкого дискретного двовимірного Фур'є-перетворення у двох режимах від розміру блоків вихідного зображення, над якими воно здійснювалось.

Таблиця 1. Таблиця залежності середнього часу виконання Фур'є-перетворення від розміру блоку.

Розмір блоку, n	Кількість блоків	CUFFT, s	FFTW, s
100	14 443 024	1,5727516378e+05	7,0830211000e+02
200	13 679 424	1,5259821477e+05	2,6701131840e+03
300	12 935 824	1,4707994075e+05	6,2489050600e+03
400	12 212 224	1,4149663598e+05	1,1190259624e+04
500	11 508 624	1,3492542941e+05	1,6189681116e+04
600	10 825 024	1,2884771038e+05	2,3923176734e+04
700	10 161 424	1,2184231721e+05	3,0840263719e+04
800	9 517 824	1,1520136774e+05	3,5017068279e+04
900	8 894 224	1,1100388477e+05	4,3881785995e+04
1000	8 290 624	9,6128184328e+04	4,9972279412e+04

Отриманні результати демонструють ефективність використання графічних процесорів для виконання громіздких неграфічних обчислень, зокрема Фур'є-перетворення, при оптимальному розмірі задачі, яка передається на графічний процесор. Оскільки, цілком очевидно, що при збільшення розміру набору даних, що опрацьовується, швидкодія графічного процесора зростає, тоді як швидкодія центрального процесора значно зменшується і, в загальному випадку, можна дійти до

рівня, коли графічний процесор буде швидше опрацьовувати дані, ніж центральний процесор, просто збільшуючи розмір набору даних, що обробляються. Різниця в порядку отриманих результатів пов'язана, насамперед, з потребою копіювання даних з пам'яті обчислювальної машини на графічний процесор. Проте ця різниця значно зменшується при збільшенні блоку даних, або при виконанні більш складної послідовності операцій обробки даних. Така тенденція пов'язана зі збільшенням паралелізму виконання операцій на графічному процесі при збільшенні об'єму даних, що опрацьовуються на графічному процесорі. Це дає змогу спростити завдання, яке ставиться перед розробниками паралельних алгоритмів. Оскільки, якщо класична реалізація обчислень ставить завдання оптимального алгоритму розбиття задачі і незалежності даних, що містяться у кожній підзадачі, то гібридна архітектура ставить завдання оптимального розміру підзадачі, що виконуватиметься на графічному процесорі, що є більш простим у порівнянні із вибором оптимального алгоритму і може бути досягнуте емпіричним шляхом без громіздкого теоретичного аналізу і чисельних оцінок швидкодії і максимального рівня паралелізму. Саме тому виконання обчислень на гібридній архітектурі кластерних обчислень дає змогу реалізувати задачі обробки великих об'ємів даних, для яких задача зводиться до вибору оптимального розміру масиву (блоку) даних, що обробляються паралельно. До таких задач належить обробка зображень високої роздільної здатності, виділення і аналіз інформативних елементів таких зображень, а також розпізнавання об'єктів складної форми за їхніми ключовими ознаками зі значно меншими затратами часу, а отже із більшою ефективністю.

Список використаної літератури

1. Chang Dar-Jen, Hierarchical Clustering with CUDA/GPU / Dar-Jen Chang, Mehmed M. Kantardzic, Ming Ouyang // ISCA PDCCS - 2009. - pp. 7-12.
2. Yang Chao-Tung, Hybrid CUDA, OpenMP, and MPI parallel programming on multicore GPU clusters / Chao-Tung Yang, Chih-Lin Huang, Cheng-Fang Lin // Computer physics communications 182, no. 1 - 2011. - pp. 266-269.
3. Yang Zhiyi, Parallel image processing based on CUDA / Zhiyi Yang, Yating Zhu, Yong Pu // Computer Science and Software Engineering, 2008 International Conference on, vol. 3 - IEEE, 2008. - pp. 198-201.
4. Gu Liang, An empirically tuned 2D and 3D FFT library on CUDA GPU / Liang Gu, Xiaoming Li, Jakob Siegel // Proceedings of the 24th ACM International Conference on Supercomputing - ACM, 2010. - pp. 305-314.
5. Nukada Akira, Auto-tuning 3-D FFT library for CUDA GPUs / Akira Nukada, Satoshi Matsuoka // Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - ACM, 2009. - p. 30.
6. Список найпотужніших суперкомп'ютерів, станом на листопад 2013 року [Електронний ресурс] — November 2013 | TOP500 Supercomputer Sites — Режим доступу до ресурсу: http://www.top500.org/lists/2015/11/.
7. Офіційна сторінка технології CUDA [Електронний ресурс] — CUDA Zone — Режим доступу до ресурсу: https://developer.nvidia.com/cuda-zone.
8. Список рішень відеокарт компанії NVIDIA з підтримкою технології CUDA [Електронний ресурс] — CUDA GPUs — Режим доступу до ресурсу: https://developer.nvidia.com/cuda-gpus.

9. Офіційна сторінка технології AMD Stream на сайті AMD [Електронний ресурс] — AMD STREAM Technology — Режим доступу до ресурсу: http://www.amd.com/en-gb/innovations/software-technologies/firepro-graphics/stream.
10. SVN-репозиторій проекту AMD CTM [Електронний ресурс] — AMD's Close-to-the-Metal / Code — Режим доступу до ресурсу: http://sourceforge.net/p/amdctm/code/HEAD/tree/.
11. Офіційна сторінка OpenCL на сайті AMD [Електронний ресурс] — OpenCL™ Zone — Режим доступу до ресурсу: http://developer.amd.com/tools-and-sdks/opencl-zone/.
12. Специфікація AMD Radeon™ HD 6990 на сайті AMD [Електронний ресурс] — AMD Radeon™ HD 6990 Graphics — Режим доступу до ресурсу: http://www.amd.com/en-us/products/graphics/desktop/6000/6990.
13. Специфікація NVIDIA Tesla K40 [Електронний ресурс] — NVIDIA® TESLA® GPU Accelerators — Режим доступу до ресурсу: http://www.nvidia.com/content/tesla/pdf/NVIDIA-Tesla-Kepler-Family-Datasheet.pdf.
14. Документація про транслювання коду мовою CUDA на мову OpenCL [Електронний ресурс] — Porting CUDA Applications to OpenCL™ — Режим доступу до ресурсу: http://developer.amd.com/tools-and-sdks/opencl-zone/opencl-resources/programming-in-opencl/porting-cuda-applications-to-opencl/.
15. NVIDIA Tesla P100 — The Most Advanced Datacenter Accelerator Ever Built Featuring Pascal GP100, the World's Fastest GPU [Електронний ресурс] — Pascal Architecture Whitepaper — Режим доступу до ресурсу: http://www.nvidia.com/object/pascal-architecture-whitepaper.html.

REALIZATION OF HYBRID ARCHITECTURE OF CLUSTER COMPUTING

V. Parubochyi, R. Shuwar

System Design Department,
Ivan Franko National University of Lviv,
50 Drahomanov St., UA–79005 Lviv, Ukraine
eddragonwolf@ukr.net

This article deals the principles of organization of hybrid architecture of cluster computing based on the using of computing opportunities of the CPU and GPU with support for GPGPU. Analyzed the advantages and disadvantages of this method of calculation, and shown the feasibility of realization of parallel programs based on the using a hybrid architecture of cluster computing. Based on the proposed programming model implemented a software framework that can be used to develop the applications. In order to demonstrate the opportunities of practical use of the hybrid architecture and to analyze its potency based on the proposed software framework, implemented program of block Fourier Transform of high-resolution images. The results of processing and substantiation and comparison of the results are given in the last part of the article.

Key words: hybrid architecture, cluster, CUDA, CTM, OpenCL, MPI, NVIDIA, GPGPU, FFTW, CUFFT.

РЕАЛИЗАЦИЯ ГИБРИДНОЙ АРХИТЕКТУРЫ КЛАСТЕРНЫХ ВЫЧИСЛЕНИЙ

В. Парубочий, Р. Шувар

Кафедра системного проектирования,
Львовский национальный университет имени Ивана Франко
ул. Драгоманова, 50, 79005, Украина
eddragonwolf@ukr.net

В статье рассматриваются принципы организации гибридной архитектуры кластерных вычислений на основе использования вычислительных возможностей центрального процессора и графического процессора, с поддержкой технологии GPGPU. Проанализированы преимущества и недостатки такого метода вычислений, а также показаны возможности реализации параллельных программ на основе использования гибридной архитектуры кластерных вычислений. На основе предложенной программной модели реализовано программный каркас, который может быть использован для разработки приложений. С целью демонстрации возможности практического использования гибридной архитектуры, а также для анализа ее эффективности, на основе предложенного программного каркаса осуществлена реализация программы блочного Фурье-преобразования изображения высокого разрешения. Результаты обработки, а также обоснование и сравнение результатов приведены в последней части статьи.

Ключевые слова: гибридная архитектура, кластер, CUDA, CTM, OpenCL, MPI, NVIDIA, GPGPU, FFTW, CUFFT.