систем створюють HPC (High Performance Computing) кластер, що дозволяє здійснити розподіл обчислювальних процедур не лише між різними процесорами, а й між різними комп'ютерами. Якщо досліджувана задача не вимагає високої взаємодії паралельних потоків, то HPC кластер може складатися з багатьох малопотужних однопроцесорних систем. Доведено, що часто такі технічні рішення забезпечують більшу продуктивність при нижчій вартості у порівнянні із продуктивністю суперкомп'ютерів [1].

Реалізація обчислювальної процедури розв'язання задачі на кластері здійснюється за посередництвом програмно створених додатків. Під паралельним додатком кластерної системи розуміють декілька процесів, що взаємодіють між собою в мережі. Якщо користувач правильно і ефективно розподілить своє завдання між процесорами на вузлах кластера, то отримає суттєвий виграш у швидкості обчислень, пропорційно числу процесорів [3].

Відзначимо, що основою для кластера є комунікаційне середовище (MPI (Message Passing Interface), PVM (Parallel Virtual Machine)), а не операційна система. Саме це середовище забезпечує ефективну взаємодію окремих частин однієї паралельної програми, що виконуються на різних комп'ютерах [1].

Для організації локальної мережі між обчислювальними вузлами кластера використовується доступна та широко розповсюджена технологія Gigabit Ethernet, адже вона як за пропускною здатністю, так і за параметрами затримок задовольняє вимоги до побудови мережі обміну по протоколу MPI [2].

IV. Висновки

Дослідження особливостей побудови кластерів показує, що при проектуванні такої системи важливим є саме вибір комунікаційного середовища, а не операційної системи. Очевидно, що для розгортання кластера, достатньо і декількох однопроцесорних систем, об'єднаних гігабітною мережею, що є значно дешевшим рішенням, ніж застосування суперкомп'ютерів. І наскільки б точним і сучасним не було обладнання, лише від правильного розподілу користувачем завдань між процесорами кластера залежатиме швидкість реалізації обчислювальної схеми.

Список використаних джерел

1. Огневий О. В. Побудова паралельних обчислювальних систем на базі кластерних технологій. // Вісник Хмельницького національного університету. – 2009. – № 4. С. 99-102.
2. Погорілий С.Д., Бойко Ю.В., Грязнов Д.Б., Ломакін О.Д., Мар'яновський В.А. Концепція створення гнучких гомогенних архітектур кластерних систем. // Проблеми програмування. – 2008. - №2-3. Спеціальний випуск. – С. 84-90.
3. http://www.intuit.ru/department/supercomputing/tbucs/class/free/status/

УДК 519.688

ДОСЛІДЖЕННЯ РОЗПАРАЛЕЛЕННЯ АЛГОРИТМУ ПОШУКУ МІНІМАЛЬНОГО ЗНАЧЕННЯ НА ГРАФІЧНОМУ ПРОЦЕСОРІ

Струбицька І.П.
Тернопільський національний економічний університет

Проблема ефективності обробки великих об'ємів даних є однією з актуальних на сьогоднішній день. Багатоядерні процесори можуть виконувати одночасно лише декілька потоків, при цьому більш висока продуктивність має іноді дуже велику ціну.

Технологія NVIDIA CUDA — це фундаментально нова архітектура обчислень на графічних процесорах, яка призначена для вирішення комплексу обчислювальних задач споживачів, бізнесу і технічної індустрії. На поточний момент обчислення на графічних процесорах з технологією CUDA — це інноваційне поєднання обчислювальних особливостей нового покоління графічних процесорів NVIDIA, які обробляють відразу тисячі потоків з високим рівнем інформаційного завантаження, які доступні через стандартну мову програмування C.

Використання GPU для обчислень загального призначення є перспективним при розв'язку задач. Наприклад, для задачі оптимізації параметрів дискретної динамічної моделі,

суть якої полягає у наступному. Побудова математичної моделі у підході [1] передбачає оптимізацію в сенсі мінімізації відхилення поведінки моделі від поведінки модельованого об'єкта, тобто мінімізація функції мети. З одного боку це дозволяє значно універсалізувати даний підхід, а з іншого – приводить до появи складних оптимізаційних задач, які важко розв'язати навіть з використанням сучасних засобів обчислювальної техніки. Задача знаходження мінімуму нелінійної функції багатьох змінних є доволі складним завданням у загальному випадку.

У задачі розпаралелення алгоритму оптимізації параметрів дискретних динамічних моделей на масивно-паралельних процесорах [1] запропоновано використати метод дрібнозернистого розпаралелення для деяких блоків алгоритму. Отже, у даному випадку можна використати паралельний алгоритм пошуку мінімального значення у масиві елементів.

Щоб оцінити наскільки добре дана технологія підходить для задач чисельного моделювання, проведемо власне тестування для порівняння часу виконання одних і тих самих програм на центральному і графічному процесорах. Усі тести були проведені на наступному апаратному забезпеченні:процесор: Core2Duo E8400, 3ГГц; графічний процесор NVIDIA GeForce GTS250, 1024 Мб, (16 мультипроцесорів по 8 процесорів).

Як приклад для тестування було вибрано задачу редукції. У загальному вигляді дана операція формулюється наступним чином. Нехай заданий масив $a_0, a_1, a_2, ..., a_{n-1}$ і деяка бінарна асоціативна операція (додавання, множення, знаходження мінімуму чи максимуму). У випадку [1] це знаходження мінімуму серед значень обчисленої функції мети.

Класичний спосіб пошуку мінімального значення в масиві елементів очевидний: припускають, що перший елемент масиву мінімальний, тоді інші елементи масиву послідовно порівнюються з цим елементом. Якщо якийсь елемент буде меншим, то він стає мінімальним значенням і продовжується перевірка. Для масиву з n елементами алгоритму потрібно $n-1$ порівнянь.

Виділимо 5 способів розпаралелення редукції на графічному процесорі [2, 3]:

1. Увесь вихідний масив розбивається на частини. Кожній частині відповідає один блок мережі (grid), який буде шукати мінімальне значення. Для розпаралелення даної операції на окремі потоки розбивають відповідні блоки елементів масиву на пари і паралельно знаходять мінімальне значення в кожній парі елементів. В результаті отримаємо вдвічі менше елементів, серед яких необхідно знайти мінімальне значення. Тоді знову розбивають їх на пари і паралельно шукають мінімум серед пари значень. Дальше знову повторюють подібний процес. Після кожного повторення число елементів буде зменшуватись вдвічі. Цей процес можна представити у вигляді рисунку 1 Основним обмеженням швидкості для такого алгоритму є доступ до пам'яті. Зручно, коли кожний блок зразу ж копіює відповідні йому елементи в shared-пам'ять і операції проводив уже у пам'яті. Також в середині циклу буде сильне розгалуження практично усіх warp'ів.

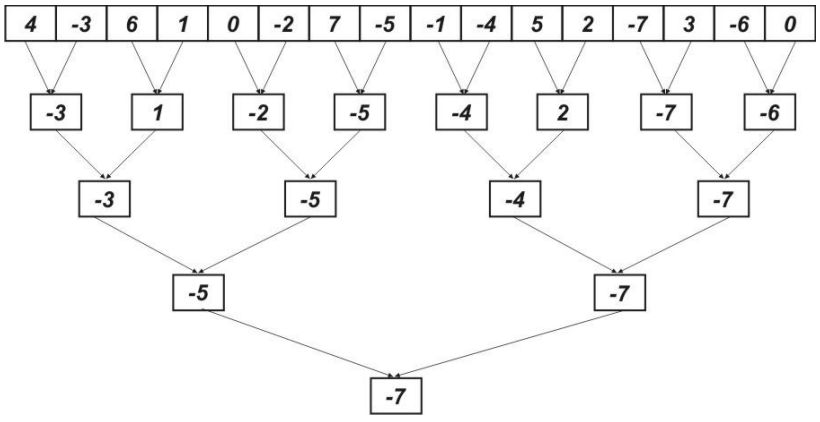

Рисунок 1 - Схема ієрархічного знаходження мінімального елементу

2. За допомогою другого способу можна уникнути розгалуження, якщо перерозподілити дані і операції по потоках. Тоді на кожному кроці циклу буде не більше одного warp'у з розгалуженням. При цьому розгалуження буде тільки для декількох останніх операцій циклу. З початку воно припадати на границю між warp'ами, тобто ніякого розгалуження не буде. Проте така реалізація має серйозний недолік: вона буде постійно приводити до конфліктів по банках високого порядку.

3. В іншому способі реалізації паралельної редукції, щоб позбутись конфліктів по банках, змінюють порядок підбору пар. В попередніх способах починають з сусідніх пар і подвоюють відстань для кожної наступної ітерації циклу. У цьому випадку — починають з пар елементів, які знаходяться на відстані $BLOCK\_SIZE/2$, і на кожному кроці зменшуємо відстань між елементами вдвоє. У цьому методі позбавились від конфліктів по банках. Але на першому кроці циклу буде загружено лише половину ниток блоку, що не є ефективним використанням можливостей GPU.

4. Якщо зменшити число блоків удвоє, то можна позбутись недоліку третього варіанту. Але при цьому кожний блок буде обробляти вдвічі більше елементів. Щоб уникнути збільшення необхідного об'єму shared-пам'яті, знаходження мінімуму у перших пар здійснюється зразу і в shared-пам'ять записується уже результат.

5. Для цієї реалізації можна зробити ще одну оптимізацію. При $s \le 32$ в кожному блоці залишиться всього по одному warp'у. Тому синхронізація буде непотрібна, як і перевірка в операторі if; цикл для $s \le 32$ можна розвернути.

За допомогою обчислювальних експериментів порівняємо швидкодію для операції редукції різними способами для $n = 2^{20} = 1048576$, де n – кількість елементів масиву (табл. 1).

Таблиця 1

Порівняння швидкодії системи для операції редукції

Варіант	Час виконання, мс	Продуктивність GPU, MFLOPS	Прискорення	Ефективність
Редукція 1 на GPU	18,90	55,48	1,64	0,013
Редукція 2 на GPU	11,04	94,98	2,81	0,021
Редукція 3 на GPU	10,45	100,34	2,97	0,023
Редукція 4 на GPU	9,21	113,85	3,37	0,026
Редукція 5 на GPU	8,10	129,45	3,83	0,029
Редукція на CPU	31,0	33,825		

У результаті даного дослідження було проведено чисельні експерименти, які показали, що при розпаралеленні методу пошуку мінімального значення продуктивність системи зростає в 3,83 рази. Отже, даний паралельний алгоритм можна використати для задачі оптимізації параметрів дискретних динамічних моделей, оскільки він зарекомендував себе як ефективний.

Але знаходження мінімуму не є самоціллю. Значення функції мети знаходяться паралельно, відсутня пересилка даних з CPU в GPU, значить практичний виграш буде ще більший.

Список використаних джерел

3. Козак Ю.Я., Стахів П.Г., Струбицька І.П. Розпаралелення алгоритму оптимізації параметрів дискретних динамічних моделей на масивно-паралельних процесорах // Відбір і обробка інформації. - 2010. – Вип. 32 (108). – С. 126-130.
4. А. В. Боресков, А. А. Харламов. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2010. – 232 с.
5. NVIDIA CUDA Compute Unified Device Architecture, Programming Guide, Version 2.0, – 2008. –107 p.