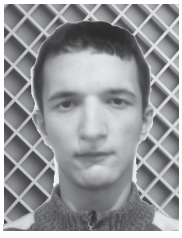
УДК 004.93'1 + 004.272 + 519.687.1

# ЗАСТОСУВАННЯ БАГАТОПОТОКОВОЇ АРХІТЕКТУРИ NVIDIA® CUDA ДЛЯ КЛАСТЕРИЗАЦІЇ ДАНИХ



Т. З. Цюра

У системі накопичення даних (Data Mining) [7] сімейство алгоритмів KNN використовується для розподілу вхідних даних по існуючих кластерах [2, 6]. Особливістю цього етапу є обробка великої кількості даних, яка здійснюється з використанням нетривіальних, ресурсоємних, проте здатних до розпаралелювання, алгоритмів (класифікація, кластеризація, розпізнавання образів). Ефективність виконання цього етапу впливає на ефективність функціонування системи в цілому [1].

**Мета статті** – застосування архітектури NVIDIA® CUDA [11] для кластеризації даних і максимізації швидкодії системи накопичення даних.

У статті порівнюються реалізації алгоритму KNN [2, 6] для виконання: на центральному процесорі (CPU); на графічному (GPU). Вони виконують кластеризацію великої кількості точок на площині, кожну з яких надано двома координатами, як метрика виступає Декартова відстань.

Роль GPU виконує графічний прискорювач NVidia GeForce 9800 GT, що містить ядро сімейства G92GT, 24 мультипроцесори, 754 млн. транзисторів і 1 Гб пам'яті GDDR3 з пропускною здатністю 56,3 Гб/с; підтримує набір команд версії 1.1. Усі GPU з даним набором команд можуть виконувати 512 синхронних потоків ($T_{max}$ = 512) на одному мультипроцесорі [8].

У ролі CPU використано AMD Phenom™ II X4 940, тактова частота якого 3 ГГц. Він містить інтегрований контролер DDR2/DDR3 пам'яті з пропускною здатністю 17,067 Гб/с [9].

Розроблені програмні реалізації алгоритму у вигляді функцій на мові C, об'єднані в одиницю компіляції, над якою виконується користувацький етап компіляції і приєднання з використанням компілятора NVCC (NVidia®). Кінцевий програмний продукт виконує виділення пам'яті для вхідних даних, ініціалізацію їх випадковими параметрами в межах існуючих кластерів, ініціалізацію таймера для оцінювання часу виконання кожної з KNN-функцій і звільнення ресурсів після виконання тесту.

Для оцінювання степені використання ресурсів графічного прискорювача використовувався програмний засіб NVidia® CUDA Visual Profiler 3.0.23 [12], з допомогою якого отримано вичерпну інформацію про виконання KNN-функцій на GPU. У випадку лінійного розподілу потоків (у кожному блоці – лише один потік, сітка одновимірна, $К_б = К_{нт}$ = A_SIZE – D_COUNT, де $К_б$ – кількість блоків, $К_{нт}$ – кількість невизначених точок) наявна велика кількість інструкцій, яка зумовлюється підвищеними витратами на ініціалізацію сітки з великою кількістю блоків і зростатиме із збільшенням $К_{нт}$.

Для оптимізації часу виконання, необхідно досягти компромісу між кількістю блоків і кількістю потоків, що має забезпечити мінімальне число інструкцій, пов'язаних з підготовкою даних до обробки і таким чином максимізувати корисну роботу GPU. Необхідне число потоків не може перевищувати максимальне число синхронних потоків ($T_{max}$) на один мультипроцесор. Це завдання виконано з допомогою програмного засобу CUDA GPU Occupancy Calculator 1.5 (рис. 1).

Отже, існує чотири альтернативні проміжки, які забезпечуватимуть повне використання ресурсів GPU. Враховуючи велику кількість звернень до глобальної пам'яті і нелінійність KNN-функції, обрано крайню ліву точку на графіку (80 потоків), яка і забезпечила мінімум (1/2) відношення кількості алгоритмічних інструкцій до кількості когерентних (uncoalesced) операцій зчитування/запису пам'яті (рис. 2).

За даної конфігурації запуску потоків (рис. 3) досягається виграш у часі виконання функції у 3–8 разів, оскільки усі 24 мультипроцесори GPU задіяні для обчислення KNN-функції.

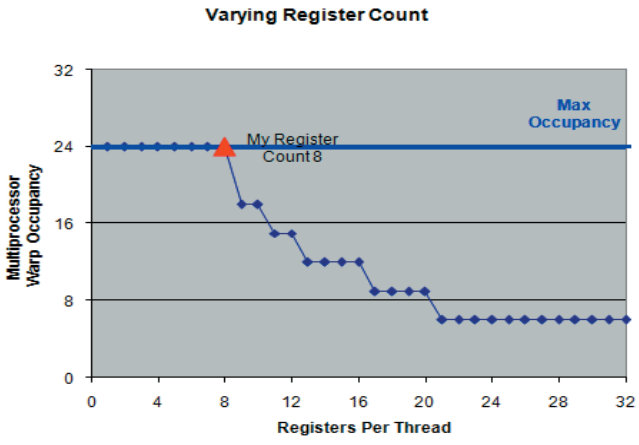
Уникнувши частого використання глобальної пам'яті, можна суттєво зменшити час виконання, оскільки звернення до глобальної пам'яті мультипроцесор виконує за 400-600 тактів, а до спільної пам'яті в блоці (__shared__) – за чотири такти [3]. Перед виконанням алгоритму кожен потік на GPU копіює відповідну (1/80) частину вхідних даних з глобальної пам'яті у спільну, після чого відбувається синхронізація в межах блоку. Обчислення здійснюються з використанням швидкої спільної пам'яті. Масив індексів точок, що беруть участь у голосуванні, компілятор також розміщає у спільній пам'яті. Усім іншим локальним змінним відповідають регістри GPU [10].

На рис. 4 показано лінійну залежність часу виконання KNN-функцій від кількості всіх точок при

**Внесення необхідних даних про GPU та про KNN-функцію**

| | |
|---|---|
| **1.) Select Compute Capability (click):** | **1,1** |
| **2.) Enter your resource usage:** | |
| Threads Per Block | 80 |
| Registers Per Thread | 8 |
| Shared Memory Per Block (bytes) | 2048 |
| **3.) GPU Occupancy Data is displayed here and in the graphs:** | |
| **Active Threads per Multiprocessor** | **640** |
| **Active Warps per Multiprocessor** | **24** |
| **Active Thread Blocks per Multiprocessor** | **8** |
| **Occupancy of each Multiprocessor** | **100%** |
| **Physical Limits for GPU:** | **1,1** |
| Threads / Warp | 32 |
| Warps / Multiprocessor | 24 |
| Threads / Multiprocessor | 768 |
| Thread Blocks / Multiprocessor | 8 |
| Total # of 32-bit registers / Multiprocessor | 8192 |
| Register allocation unit size | 256 |
| Shared Memory / Multiprocessor (bytes) | 16384 |
| Warp allocation granularity (for register allocation) | 2 |

**Ефективність використання регістрів**




**Залежність використання мультипроцесорів від кількості потоків у блоці**

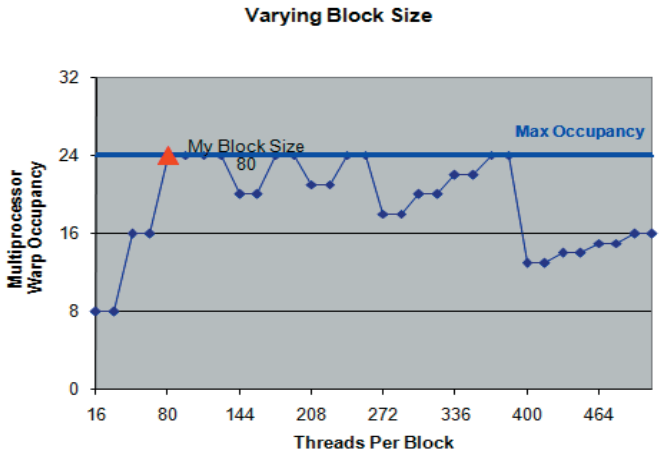



**Рис. 1. Пошук точки оптимуму**

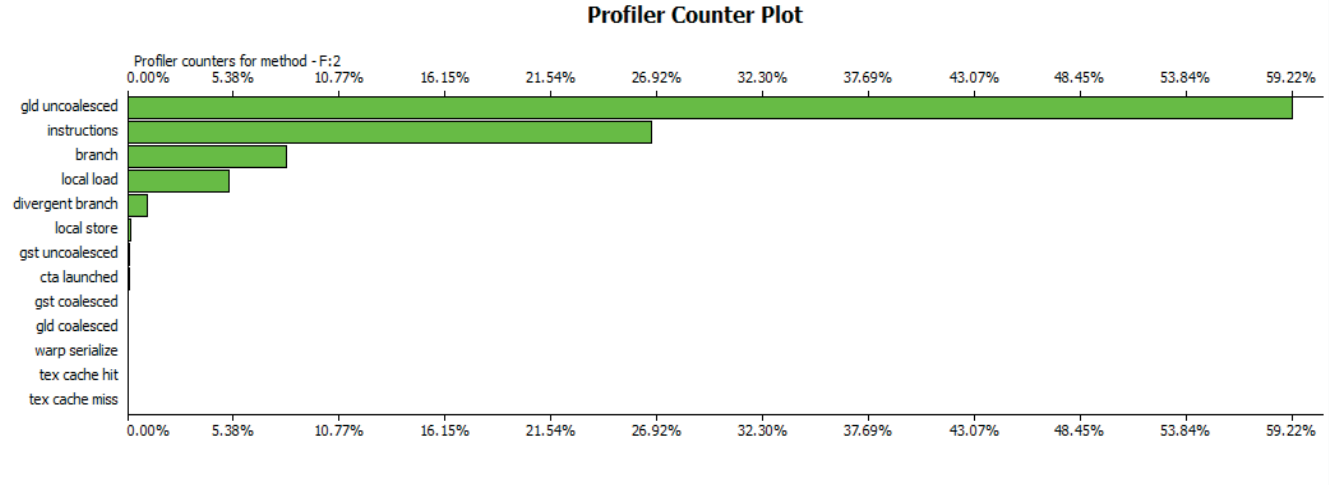



**Рис. 2. Процентний склад виконуваних інструкцій**

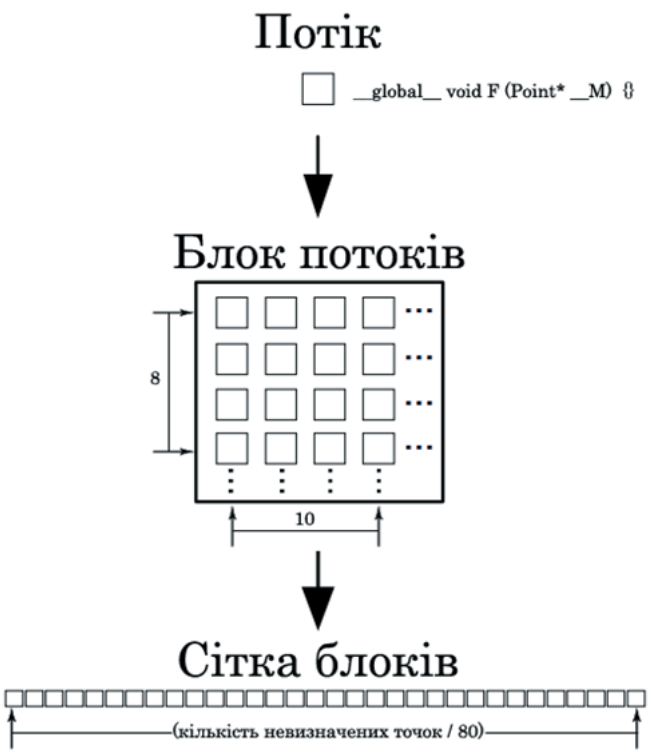



**Рис. 3. Структура розпаралелювання для GPU з 80 потоками в блоці**

фіксованому значенні інших параметрів.

Відношення кількості розподілених по кластерах точок (I_COUNT) і значення параметра алгоритму (K) до часу виконання характеризує експоненційна залежність. Це відбувається за рахунок вкладених циклів, де ітератор $\in [0..K)$.

На початку проміжку за наявності малих обсягів обчислень витрати на ініціалізацію потоків на GPU співмірні з витратами на виконання власне алгоритму, що різко збільшує час виконання KNN-функції відносно часу CPU та нівелює доцільність розпаралелювання. У разі збільшення обсягів вхідних даних швидко зростає відношення часу виконання CPU до часу GPU, проходячи точку при кількості активних потоків, що дорівнює 80, зростаючи все швидше на подальшому проміжку. На кінцевих точках виграш у швидкодії $\frac{t(CPU)}{t(GPU)} \approx 30-70$ разів.

Залежність на графіку має слабкі флуктуації, що чіткіше виражені для GPU і посилюються зі збільшенням загальної кількості точок (A_COUNT). Вони виникають через особливості розподілу інструкцій для виконання в багатопоточних системах [5].

На CPU-архітектурі підтримується дійсна (за

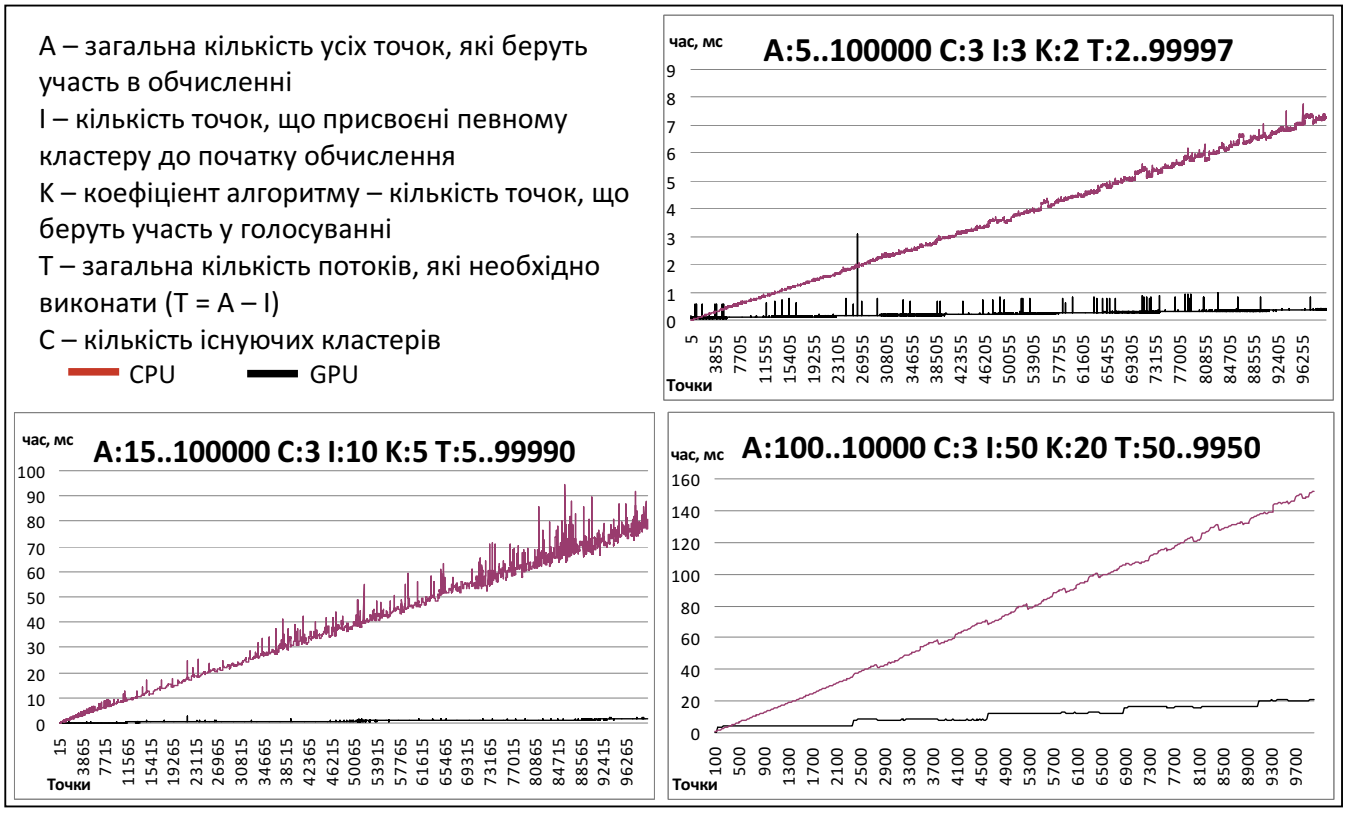



**Рис. 4. Результати експерименту**

умови наявності декількох ядер і/або процесорів) і псевдобагатопоточність (черга команд для одного ядра/процесора). Ядро, яке виконує набір інструкцій одного процесу, може проявляти псевдобагатопоточність і перемикатися на виконання набору інструкцій іншого процесу. У цьому випадку операційна система ініціюватиме чергу команд і додасть до неї команди збереження і відновлення регістрів процесора, які потребуватимуть додаткового часу виконання.

На обох досліджуваних архітектурах не гарантується ексклюзивне виконання одного набору інструкцій на окремій обчислювальній одиниці [1]. Виконання KNN-функції на деяких мультипроцесорах GPU призупиняється для виконання графічних обчислень (наприклад, роботи відеодрайвера). Пізніше ці мультипроцесори завершують роботу із затримкою, що дорівнює тривалості виконання графічних обчислень. Також наявна інструментальна похибка вимірювання часу виконання. Її відносну величину можна знайти за формулою:

$$\delta = \frac{\Delta}{t} * 100\%,$$

де $\Delta$ – час виконання cutStartTimer() і cutGetTimerValue() з модуля cutil_inline.h [4];

t – загальний час виконання KNN-функції. Для точки $\frac{t(CPU)}{t(GPU)} \approx 1, \delta \approx 0{,}001\%$ і продовжує спадати на подальшому проміжку.

**Висновки**

Виконано проектування, реалізацію і тестування роботи алгоритму KNN на програмно-апаратній архітектурі NVidia® CUDA мовою C++ з використанням стандартних бібліотек C і бібліотек NVidia® CUDA SDK.

Під час проектування застосовано:

- методи дослідження операцій для пошуку точки оптимуму ефективності (80 активних потоків);
- теорію побудови багатопоточних систем для дослідження природи похибок.
- чисельні методи для детермінування значень похибок.

За результатами тестування програмної реалізації KNN досліджено залежності між обсягом вхідних даних і часом роботи алгоритму, підтверджено доцільність розпаралелювання і високу ефективність GPU як невід'ємної частини сучасної системи накопичення даних.

ЛІТЕРАТУРА


1. *Вальковский, В. А.* Распараллеливание алгоритмов и программ: Структур.подход / Вальковский В.А. – М. : Радио и связь, 1989. – 175с. – ISBN 5256001957.

2. Top 10 algorithms in data mining / [Xindong Wu, Vipin Kumar , J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, Dan Steinberg] – USA, New York: Springer – Verlag New York, Inc., 2007. –37 с. – Бібліогр.: с. 1 – 37. – ISSN:0219-1377.

3. NVIDIA Corp.: OpenCL Optimization [Електронний ресурс]: 2010. – 31 р. – Режим доступу: http://developer.download.nvidia.com/CUDA/training/NVIDIA_GPU_Computing_Webinars_Best_Practises_For_OpenCL_Programming.pdf. – Назва з екрана.

4. NVIDIA Corp.: CUDA C Programming Guide [Електронний ресурс]: 2010. – Режим доступу: http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Programming_Guide.pdf . – Назва з екрана.

5. NVIDIA Corp.: CUDA Reference Manual [Електронний ресурс]: 2010. – Режим доступу: http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_Toolkit_Reference_Manual.pdf . – Назва з екрана.

6. *Сергей Царьков*: Алгоритм ближайшего соседа [Електронний ресурс]: BaseGroup Labs. – 1995 – 2010. – Режим доступу: http://www.basegroup.ru/library/analysis/regression/knn/. – Назва з екрана.

7. *Николай Паклин*: Алгоритмы кластеризации на службе Data Mining [Електронний ресурс]: BaseGroup Labs. – 1995 – 2010. – Режим доступу: http://www.basegroup.ru/library/analysis/clusterization/datamining/. – Назва з екрана.

8. NVIDIA Corp.: NVIDIA GeForce 9800 GT: supercharge your graphics horsepower with this GPU at the perfect price and performance combination [Електронний ресурс]: 2010. – Режим доступу: http://www.nvidia.com/object/product_geforce_9800gt_us.html. – Назва з екрана.

9. AMD: AMD Phenom™ II Processors [Електронний ресурс]: 2010. – Режим доступу: http://www.amd.com/us/products/desktop/processors/phenom-ii/Pages/phenom-ii.aspx. – Назва з екрана.

10. NVIDIA Corp.: CUDA C Best Practices Guide [Електронний ресурс]: 2010. – 20 р. – Режим доступу: http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/CUDA_C_Best_Practices_Guide.pdf. – Назва з екрана.

11. NVIDIA Corp.: NVIDIA GPU Computing Developer Home Page [Електронний ресурс]: 2010. – Режим доступу: http://developer.nvidia.com/object/gpucomputing.html. – Назва з екрана.

12. NVIDIA Corp.: NVIDIA Compute Visual Profiler Version 3.2 [Електронний ресурс]: 2010. – Режим доступу: http://developer.download.nvidia.com/compute/cuda/3_2/toolkit/docs/VisualProfiler/computeprof.html. – Назва з екрана.