一個以繪圖處理器為基礎之記憶體資料庫實

劉嘉倩
交通大學資訊科學與工程研究所
chiam.bes@gmail.com

徐竣傑
交通大學資訊科學與工程研究所
cchsu77@gmail.com

***摘要*—近年來，NVIDIA 致力於 GPGPU 的發展，一個高度平行化的發展平台 CUDA 就此產生。使用者利用熟悉的 C 語言就可以在上面開發自己的應用程式，加上記憶體空間的快速成長，已經足夠一個資料庫的使用。因此，我們在 GPU 的記憶體上面實作了一個實驗性的資料庫，並觀察 GPU 的計算能力，如何改善一般資料庫的操作效能。**

***關鍵詞*— GPU; CUDA; Turning point**

一、簡介

1.1 前言

近年來，由於網站的大量增加和電子商務的湧現，對於資料應用的需求越來越大，進而產生許多的資料庫。資料庫管理系統可用於收集訊息和提供查詢、更新和刪除資料的功能。目前，從手機到網頁伺服器皆使用資料庫來管理其所需的資料和訊息。

1.2 動機

資料庫管理系統的效能是極為重要的。以一個擁有數千名會員的中型網站而言，客戶端在查詢資料時，系統的回應時間勢必越短越好。此外，GPGPU的軟體開發套件已經非常方便，使用它來設計演算法和應用程式可以得到更好的效能。總和以上兩點，使用GPGPU來實作一個資料庫系統將是一個有趣的議題。

1.3 問題描述

我們選擇CUDA為平台，然而使用CUDA來開發應用程式會有幾個GPU硬體上的限制：

(1) 暫存器和共享記憶體數量上的限制
(2) 條件判斷式對於效能的影響
(3) CPU和GPU間資料傳遞的負擔
(4) 不連續的記憶體存取

對於(1)，我們讓每個執行緒使用到暫存器和共享記憶體越少越好，以期提升平行運算的能力。對於(2)，我們將部分的條件判斷交由CPU來處理，我們將整個資料庫的內容儲存在GPU的記憶體中，以減少(3)所造成的影響。對於(4)，我們將資料以column major的方式存放於資料表中，來達成coalesced memory access。

1.4 研究目的

在GPU上實做數個常用的函式，在GPU DB上進行時間的測量，並與SQLite memory DB進行效能上的觀察和比較。

1.5 研究貢獻

根據我們實驗的結果，此二種資料庫的效能會因為資料量的上升而產生一個turning point。turning point代表被查詢的資料和總共資料的一個比值，我們測試每個函式以取得turning point的資訊，繼而得到一個結論：當兩者間的比值超過turning point時，本論文所實作之資料庫比起SQLite memory DB會有更佳的效能

二、背景及相關文獻

2.1 圖形處理器(GPU)

圖形處理器是一個特殊功能的處理器，具有大量平行化運算能力，負責支援CPU處理3D圖形。理論上任何能對應成串流處理的工作GPU都能勝任。由於硬體技術的不斷進步，GPU不只是單純的圖形處理器，它可以處理許多CPU所執行的一般性的應用。

CPU和GPU之最主要的不同在於，GPU是特別針對圖形處理，具有高度平行化運算能力。相較於資料條件判斷和資料的快取，GPU專注於大量平行的資料處理能力，如圖2-1。

Figure 2-1 Layout of CPU and GPU[11]

目前有兩個最大的GPU公司，NVIDIA和AMD/ATI，致力於提供軟體開發的介面，CUDA和CTM，讓軟體設計人員能更易於開發。

2.2 CUDA by NVIDIA

CUDA(Compute Unified Device Architecture)是一個包含新的平行化程式模型和指令集架構的一般目的的平行運算架構。CUDA使用現在廣為流行的C語言來撰寫利用大量執行緒來達成平行化計算的程式[6][8]，其原始碼是由host code(在CPU上執行的程式碼)和device code(在GPU上執行的程式碼)所組成[11]。

2.3 Close To Metal (CTM) by AMD/ATI

CTM是由AMD/ATI所開發平行化程式的架構，相較於CUDA，CTM顯得更為低階，類似於組合語言。要將CTM對應到一般的應用上比較困難，所以相較於CUDA，CTM較不普遍。

2.4 主記憶體資料庫

主記憶體資料庫是將資料放在主記憶體中的資料庫管理系統。由於硬體技術的不斷進步，記憶體變的越來越便宜，容量也越來越大，要將資料庫存放在主記憶體中已非不可能的任務[19][20][21][22][23]。相較於將資料存放在硬碟內的系統，主記憶體資料庫有著兩個特性：

(1) 存取的時間較短

(2) 記憶體通常是具揮發性的

2.5 相關文獻

2.5.1 Implementing Database Operation Using SIMD Instructions

2002年的論文”Implementing Database Operation Using SIMD Instructions”使用SIMD指令實作了數個資料庫運算的演算法，在這些運算中，運算元都是使用128-bit的暫存器，每個來源運算元包含四個32-bit的單精度浮點數，而目的運算元儲存各平行算的結果。這篇論文顯示使用此SIMD指令所的演算法比起傳統的演算法的執行時間有著10%到4倍的減少。

2.5.2 Fast Computation of Database Operations using Graphics Processors

2004年的論文，”Fast Computation of Database Operations using Graphics Processors”[31]在NVIDIA GeForce FX 5900上實作數個資料庫的運算(未使用CUDA)。結果顯示此實作比傳統CPU的演算法快兩倍。

三、在GPU的記憶體上實作一個資料庫

我們列出數個在一般資料庫上常見的函式，在我們的系統上加以實作。在後面的段落中，我們將顯示卡稱為device，將主機稱為Host。

3.1 資料庫的架構

圖3-1列出我們在圖形處理器上實做的資料庫架構。初始時，所有的資料和表格會由Host 主記憶體讀入device的記憶體，而後便釋放主記憶體，在我們的系統中，主記憶體只負責記錄表格的信息和指標。由於GPU和CPU之間的溝通是透過PCI-Express 匯流排，要將資料在兩個處理器間傳遞的代價是非常昂貴的，要盡量避免。

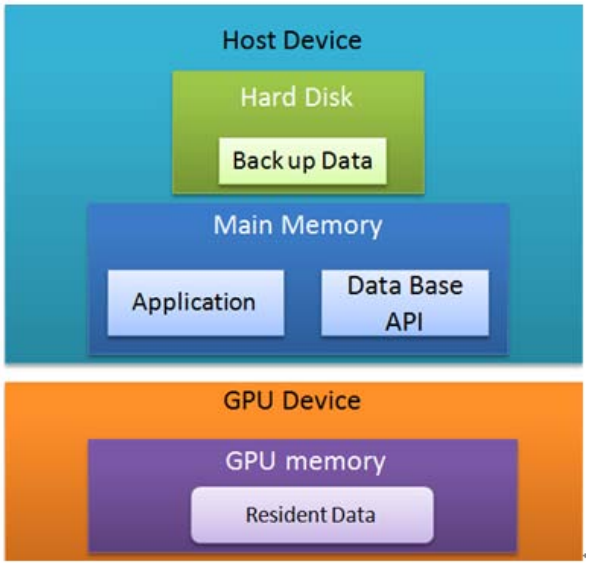

Figure 3-1 Architecture of entire system

3.2 *資料庫函式*

3.2.1 資料結構

一般資料庫使用tree的結構來管理和儲存資料，然而B-tree的搜尋演算並不適合平行計算的架構，因此，我們使用陣列來做為資料結構，由於陣列支援任意存取，更加適合平行運算，此外，由於GPU是row-major，依照row-major的順序來排序資料很難達成coalesced memory access。為避免這個問題，表格以column-major的形式存在設備記憶體當中。

3.2.2 Selection Query

在我們的系統中，Selection query的平行化實作方式，在一個表格中每個執行緒負責一個column，根據條件來判斷，平行的執行比較，另外用一個陣列來記錄結果，若該column是要取的，在陣列中相對應的值即為1否則為0。

事實上，存在著兩個問題：

(1)"And"和"OR"的邏輯運算優先權不一樣

(2)條件判斷式會造成效能上的影響

為解決上述的問題，我們使用後序來代替前序，而並將部分條件判斷的處理交由HOST來降低效能上的損失。Query的結果必須要存放到一個二維陣列，我們使用CUDPP所提供的函式"cudppScan"執行prefix sum來計算出正確的位置。圖3-2為Selection query的流程圖。

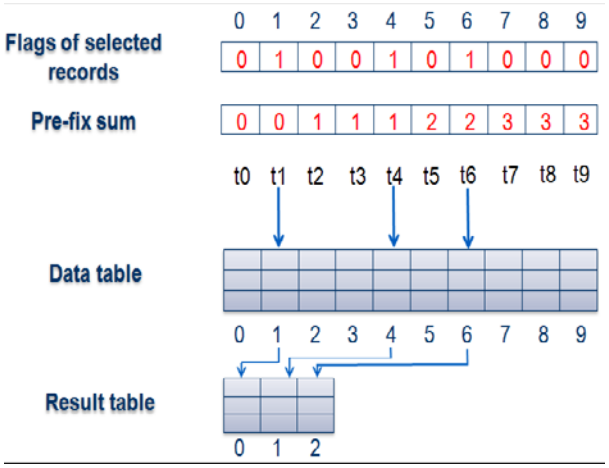

Figure 3-2 Process of moving data queried to the result table

3.2.3 Sorting Data

我們的排序函式是修改CUDPP的排序演算法而來的，輸入兩個一維陣列，一個存資料一個存位排序前各資料的初始位置，輸出為兩個一維陣列，一個存序後的資料，一個存排序後資料對應的初始位置。圖3-3為排序的範例。

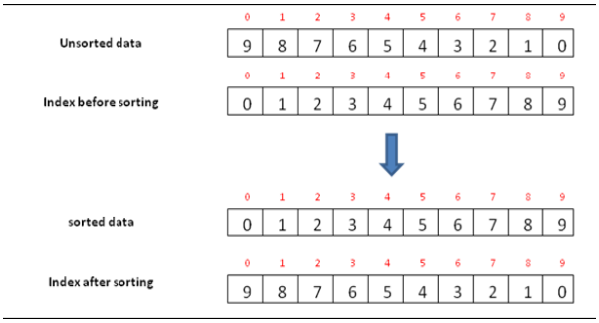

Figure 3 - 3 Modified algorithm of parallel sorting

3.2.4 Data Grouping

首先要將表格中的資料依不同條件分群，將相同群的column放在一起，在使用CUDPP中的cudppSegmentationScan將總和加起來(圖3-4)。

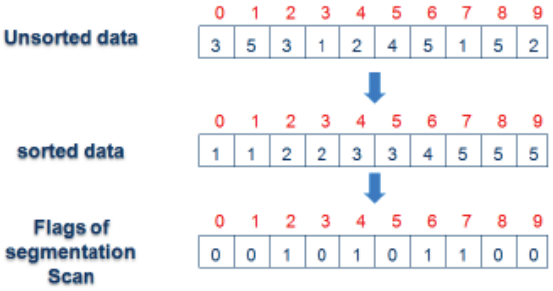

Figure 3 - 4 Setting flags of segmentation Scan

四、實驗及分析

4.1 實驗配置

4.1.1 A Memory Data base – SQLite

SQLite是一種遵守ACID的關聯式資料庫管理系統，被包含於C的函式庫中。由於在消耗總量、延遲時間和整體簡單性上，SQLite都有著不錯的表現，因此許多應用程式皆使用SQLite，例如：Mozilla Firefox，Mac computer，iPhones，等等。SQLite有兩種模式：disk mode and memory mode，在此實驗中，我們以相同功能的函式來評估GPU DB和memory mode下的SQLite兩者的效能。

4.1.2 Hardware Configuration

表4-1為我們所使用的硬體資訊。

Table 4 - 1 Hardware configuration

CPU	Intel Core 2 Quad Q6600 (2.4GHz, four core)
Motherboard	ASUS P5E-VM-DO-BP, Intel® X38 Chipset
RAM	Transcend DDR-800 2G
GPU	NVIDIA 9800 GT 512MB (GIGABYTE OEM)
HDD	WD 250G w/ 8MB buffer

由於我們的資料庫是建立在GPU的記憶體上，表4-2為GPU device的資訊。

Table 4 - 2 NVIDIA 9800GT Hardware Specification

Core Name	GeForce 9800 GT (G92)
Number of Multi-Processor	16
Number of Registers	8192 (per SIMD processor)
Constant Cache	8KB (per SIMD processor)
Texture Cache	8KB (per SIMD processor)
Processor Clock Frequency	Shader: 1.751 GHz, Core: 700 MHz
Memory Clock Frequency	900 MHz
Shared Memory Size	16KB (per SIMD processor)
Device Memory Size	512MB GDDR3

4.1.3 Software Configuration

Table 4 - 3 Software Configuration

OS	Open SUSE with version 11.1 (32bit version)
GPU Driver Version	185.18.14
CUDA Version	2.2
GNU Compiler	gcc41

4.2 評估和分析

由於數個函式的比較結果都非常類似，我們取Selection Query, Data Grouping和Insert Data來比較GPU DB和SQLite的效能，接著討論資料總數和查詢結果的數目對於GPU執行時間的關係。

4.2.1 測試資料

每個表格都有含有三個column，column1的值表此資料是此表格內的第幾筆，column2儲存的值表此資料是屬於哪個group，column3內儲存

一個介於1到65535的整數。

Column 1	Column 2	Column 3
2	50	598

4.2.2 函式的效能評估

4.2.2.1 Insertion

在實驗中，GPU DB和SQLite相比，兩者差距只在20ms到120ms之間。其中，測試資料每增加500筆，GPU和CPU資料傳輸平均增加10.452ms，GPU DB的平均執行時間就增加65.512ms。

4.2.2.2 Selection Query

在資料量小於2500，GPU DB執行Selection Query平均需要2.598ms，當測試資料少於1700筆時，SQLite memory DB的效能會優於GPU DB，當資料量大於1700到10,000，GPU DB總是有較佳的效能。(附錄二)

4.2.2.3 Data Grouping

Data Grouping包含兩個步驟，查詢和排序，我們經由計算在GPU上執行的核心程式的時間來分析其效能。

Data Grouping和Selection Query相似，在查詢資料界於2500筆到10,000筆時，我們的DB總是有較好的效能。

4.3 Turning point的評估

Turning point的出現代表著我們的GPU DB的效能超越了SQLite memory DB，它是效能上的轉折點，圖4-1列出各函式之Turning point在總資料和查詢資料量上的關係，縱軸代表被查詢的資料數目，而橫軸則是整個資料表的大小。

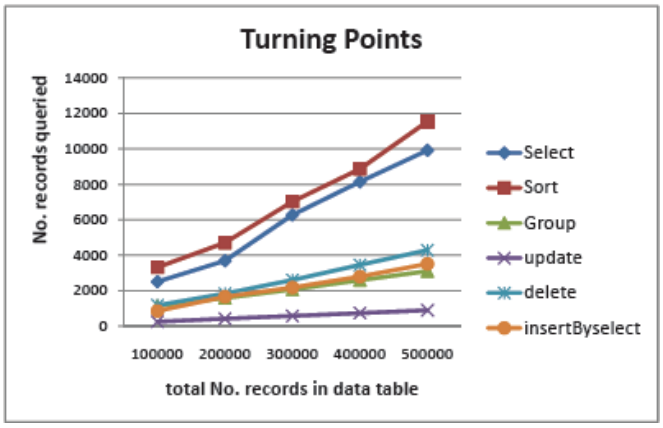

Figure 4-1 Turning points of GPU DB and SQLite memory DB

表4-1是根據圖4-1所產生，當被查詢的資料超過總資料量的0.161%到2.061%時，我們的資料庫總是有較好的效能。

Function Name	Ratio of No. data queried to total No. records (%)
Selection Query	1.926%
Selection Query and Sorting Data	2.061%
Selection Query and Data Grouping operations	0.491%
Data Insert According to Selection Query	0.784%
Data Update	0.161%
Data Delete	0.784%

Table 4 – 1 the ratios of each function

五、結論和未來規劃

5.1 結論

使用GPU來處理需要高密度計算的平行性問題，通常可以獲得令人驚艷的效能改善，尤其近年來，GPGPU的發展已經提升了許多需要在主要處理器上做運算的應用程式的處理速度。在此篇論文中，我們調查了目前已存在的主要記憶體數據庫與CUDA程式模組的一些背景，並提出了一個整體性的全新架構，主要的數據庫操作包含

資料排序、字首和程序以及聚集函數演算。而這些運算，例如選取的查詢操作，對於記憶體運算與輸入/輸出運算的比例而言是很小的。

實驗結果顯示：我們的GPU數據庫在各個資料表的運算所花費的執行時間，有著與相同運算之所有資料表紀錄幾乎一樣的執行時間，也就是說，這些我們所謂資料操作的執行時間，並非與資料表中有多少筆數的資料紀錄的詢問操作有著絕對密切的關係。這實驗結果顯示了我們的GPU數據庫與SQLite數據庫之間的差異。現有SQLite數據庫的執行時間會與資料數成正比，越多的記錄筆數花費越多的執行時間，但反觀我們的GPU數據庫，執行時間所受u影響的因素並非每個查詢操作的資料表有多少筆紀錄，反而是總共有多少筆紀錄在資料表中。基於此種特性，我們評估了我們的GPU數據庫與現有SQLite數據庫間的轉折點，此轉折點的變化趨勢為一線性規則，因此，我們針對所有記錄的查詢操作推算出一個近似比值。最後，我們得到了一個很小的比例值，依據不同的操作函數，大約0.161%到2.061%的結果值。並且，我們相信在一般的常見例子中，這些值是很容易被超過的。

5.2 未來規劃

看完我們所提出的GPU數據庫之功能改進後，我們仍為進一步的加強考慮了許多的方向。

5.2.1 平行字串的資料查詢

由於平行字串的資料查詢目前並無適合的解決方法，我們的GPU數據庫目前是僅支援數字格式，字串格式的資料必須被儲存成數字格式或是其他類似二維陣列等其他資料格式才能處理。因為所有的字母皆為字元排序，所以比對字母應該可以從字首到字尾去比較。但為了要平行比對，這種循序的字串特徵似乎是未克服的。然而，此項功能未來勢必是需要被完全實作出的。

5.2.2 關聯查詢

我們的GPU數據庫目前並無法支援關聯查詢。在外來規劃中，我們將會設計一個關聯資料表去管理各資料表間的關係。

5.2.3 協同資料庫查詢

因為我們的GPU數據庫在同一時間僅能處理一個全域的函數呼叫，所以我們計畫去設計一個排程器。這個排程器可以合併多個請求成一個GPU函數呼叫，且在一個多數使用者的環境中，對於協同資料庫的查詢維持資料一致性。

參考資料

[1] Tobin J. Lehman and Michael J. Carey, “A Study of Index Structures for Main MemoryDatabase Management Systems”, VLDB, Kyoto, Japan, 1986.

[2] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong and Tor M.Aamodt,“Analyzing CUDA workloads using a detailed GPU simulator”, Performance Analysis of Systems and Software-ISPASS, Boston, Massachusetts, 2009.

[3] Shubhabrata Sengupta, Mark Harris, Yao Zhang, and John D. Owens, “Scan Primitives for GPU Computing”, ACM, 2007.

[4] Mark Harris, “Parallel Prefix Sum(Scan) with CUDA”, NVIDIA, 2008.

[5] Garland M, Le Grand S, Nickolls J, Anderson J, Hardwick J, Morton S, Phillips E, Yao Zhang, Volkov V, “Parallel Computing Experiences with CUDA”. IEEE, Los Alamitos, CA, USA, 2008.

[6] Qihang Huang, Zhiyi Huang, Paul Werstein, and Martin Purvis, “GPU as a General Purpose Computing Resource”. IEEE , Washington, DC, USA, 2008.

[7] Ziyi Liu, Wenjing Ma, “Exploiting Computing Power on Graphics Processing Unit” IEEE, Washington, DC, USA, 2008.

[8] David Luebke, “CUDA: Scalable Parallel Programming for High-Performance Scientific Computing”, NVIDIA 2008.

[9] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A Unified Graphics and Computing Architecture”, IEEE, 2008.

[10] “NVIDIA CUDA Programming Guide, 2.0 edition”, NVIDIA Corporation 2008.

[11] “NVIDIA CUDA Programming Guide, 2.2 edition”, NVIDIA Corporation 2009.

[12] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Mike, and H. Pat, “Brook for gpus: Stream computing on graphics hardware”, ACM, New York, USA, 2004.

[13] Shin-Jae Lee, Minsoo Jeon, Andrew Sohn, and Dongseung Kim, “Partitioned Parallel Radix Sort”, ISHPC, Tokyo, Japan, 2000.

[14] Pablo Barcel´o, “Logical Foundations of Relational Data Exchange,ACM,SIGMOD, Mrch”, New York, US, 2009.

[15] T. Purcell, I. Buck, W. Mark, and P. Hanrahan, “Ray tracing on programmable graphics hardware. ACM Trans on Graphics”, SIGGRAPH, San Antonio, Texas USA, 2002.

[16] T. Purcell, C. Donner, M. Cammarano, H. Jensen, and P. Hanrahan, “Photon mapping on programmable graphics hardware”, SIGGRAPH/Eurographics Conference on Graphics Hardware, San Diego, California, USA, 2003.

[17] E. S. Larsen and D. K. McAllister. “Fast matrix multiplies using graphics hardware”, IEEE Supercomputing, 2001.

[18] D. Manocha, “Interactive Geometric and Scientific Computations using Graphics Hardware”, SIGGRAPH, 2003.

[19] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. “Weaving relations for cache performance”, In Proceedings of the Twenty-seventh International Conference on Very Large Data Bases, 2001.

[20] S. Manegold, P. Boncz, and M L. Kersten, “What happens during a join? Dissecting CPU and memory optimization effects”, VLDB, Proceedings of 26th International Conference on Very Large Data Bases, Cairo, Egypt, , September 10-14, 2000.

[21] Shintaro Meki and Yahiko Kambayashi. “Acceleration of relational database operations on vector processors”. Systems and Computers, Japan, August 2000.

[22] Jun Rao and Kenneth A. Ross. “Cache conscious indexing for decision-support in main memory”, VLDB, 1999.

[23] Kenneth A. Ross. “Conjunctive selection conditions in main memory”, Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems: PODS, 2002.

[24] M. Macedonia. , “The gpu enters computing's mainstream”. IEEE, October 2003.

[25] Honghoon Jang; Anjin Park; Keechul Jung, “Neural Network Implementation Using CUDA and OpenMP”, Computing: Techniques and Applications, 2008.

[26] Guobin Shen, Lihua Zhu, Shipeng Li, Heung-Yeung Shum, Ya-Qin Zhang, “Accelerating video decoding using GPU”**,** IEEE International Conference,2003.

[27] Kenneth Moreland, Edward Angel, “The FFT on a GPU”, Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, San Diego, California, 2003.

[28] Simek, V, Asn, R.R, “GPU Acceleration of 2D-DWT Image Compression in MATLAB with CUDA”**,** Computer Modeling and Simulation, 2008.

[29] Wei-Nien Chen; Hsueh-Ming Hang, “H.264/AVC motion estimation implmentation on Compute Unified Device Architecture (CUDA)”, IEEE International Conference 2008.

[30] Manavski, S.A, “CUDA Compatible GPU as an Efficient Hardware Accelerator for AES Cryptograph”**,** ICSPC 2007.

[31] Naga K. Govindaraju , Brandon Lloyd , Wei Wang , Ming Lin , Dinesh Manocha, “Fast computation of database operations using graphics processors”, ACM SIGMODm international conference on Management of data, Paris, France, June 13-18, 2004.

[32] Jingren Zhou, Kenneth A. Ross, “Implementing Database Operations Using SIMD Instructions”, ACM SIGMOD, Madison, Wisconsin, USA, 2001.