Data Mining 2018

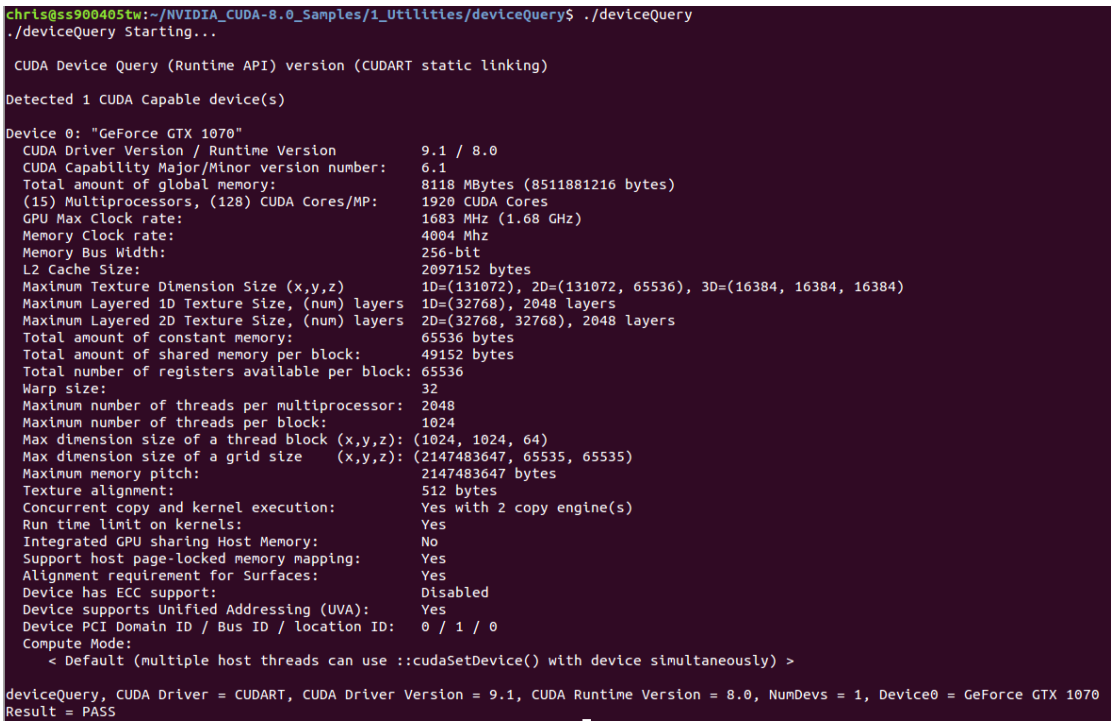
HW2 Frequent Itemset Mining on GPGPU

姓名：吳睿哲

學號 ：r06921095

日期 ： 2018/11/03

Part I. Setting up CUDA environment



```
chris@ss900405tw:~/NVIDIA_CUDA-8.0_Samples/1_Utilities/deviceQuery$ ./deviceQuery
./deviceQuery Starting...

 CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce GTX 1070"
  CUDA Driver Version / Runtime Version          9.1 / 8.0
  CUDA Capability Major/Minor version number:    6.1
  Total amount of global memory:                 8118 MBytes (8511881216 bytes)
  (15) Multiprocessors, (128) CUDA Cores/MP:     1920 CUDA Cores
  GPU Max Clock rate:                            1683 MHz (1.68 GHz)
  Memory Clock rate:                             4004 Mhz
  Memory Bus Width:                              256-bit
  L2 Cache Size:                                 2097152 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(131072), 2D=(131072, 65536), 3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers  1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(32768, 32768), 2048 layers
  Total amount of constant memory:               65536 bytes
  Total amount of shared memory per block:       49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                     32
  Maximum number of threads per multiprocessor:  2048
  Maximum number of threads per block:           1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                             512 bytes
  Concurrent copy and kernel execution:          Yes with 2 copy engine(s)
  Run time limit on kernels:                     Yes
  Integrated GPU sharing Host Memory:            No
  Support host page-locked memory mapping:       Yes
  Alignment requirement for Surfaces:            Yes
  Device has ECC support:                        Disabled
  Device supports Unified Addressing (UVA):      Yes
  Device PCI Domain ID / Bus ID / location ID:   0 / 1 / 0
  Compute Mode:
     < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 9.1, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GeForce GTX 1070
Result = PASS
```

Part II. Frequent Itemset Mining with GPGPU

Code 的部分我是直接使用助教提供的 sample code，直接修改

function :mineGPU()，並增加了兩個在 gpu 中執行的 function:

__global__ static void vecIntersect()、

__device__ int NumberOfSetBits_dev()。

在 mineGPU 中:我主要是在做向量內積時做做平行化處理，把兩個要做內積的向量分別 copy 到 device，但是這樣的作法會花費許多時間在做 malloc 跟 memory copy，所以我盡量把這些動作都移到 for 迴圈的外面，這樣的做法可以讓 memory copy 到 device 的次數減少，花費的時間也會大幅減少。

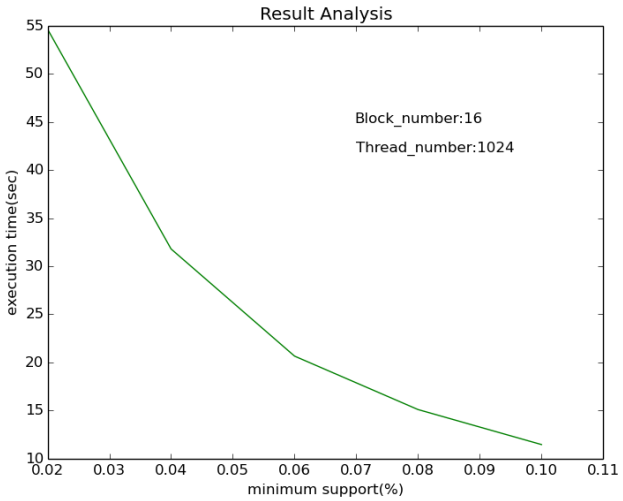
再來是 device 端執行的部分: __global__ static void vecIntersect()，這個地方我花了比較多工夫，首先我使用了 20 個 block，每個 block 中有 1024 個 thread，這個地方可以用每個 block 中的 thread 去做連續記憶體中的存取，以減少從 global memory 讀取資料時的 latency。所以 thread number 不能取得太小。

另外，還可以使用 shared_memory 去儲存兩條向量內積的結果， shared memory 是一個 block 中每個 thread 都共用的記憶體，它會使用在 GPU 上的記憶體，所以存取的速度相當快，不需要擔心 latency 的問題。

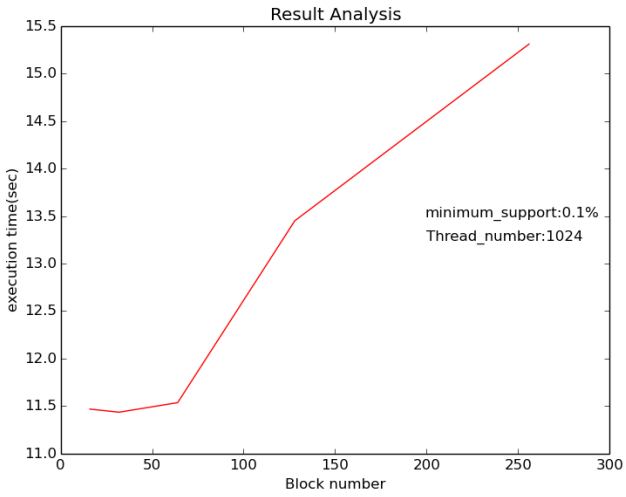
最後是把每一個 block 中的 thread 做加總的部分，這邊我是參考這篇文章: http://www2.kimicat.com/%E6%94%B9%E8%89%AF%E7%AC%AC%E4%B8%80%E5%80%8B%E7%A8%8B%E5%BC%8F，使用樹狀加法，然後再把相加的結果傳回 host 端，把每個 block 的 support number 相加即可達到最終的 support number。

我的演算法 gpu 的執行結果平均比 cpu 快 1.7 倍，感覺可以使用 2D 的 Memory Copy 直接把所有的 bit vector 一次傳到 device 裡面，應該會比這個版本快很多，但由於時間問題，所以先做到這樣。

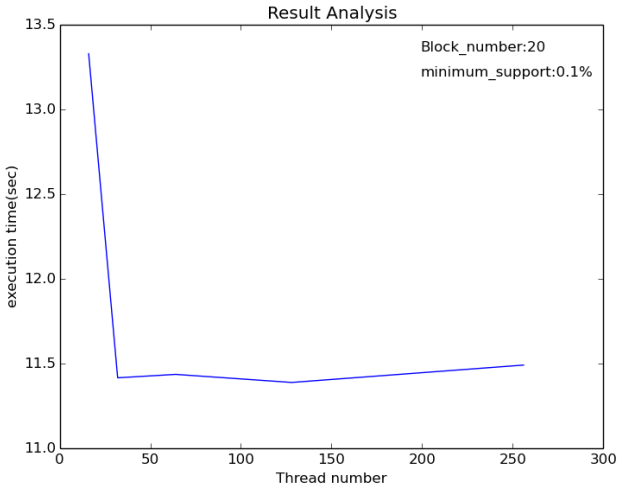
Plot



這張圖使用 16 個 block，1024 個 thread number 得到的結果，花費的時間會應取的 minimum support 越小，變慢的幅度越大。



這張圖是使用 0.1%的 minimum support，1024 個 thread number 得到的結果，隨著 block number 取得越大花費的時間會越多，我覺得可能跟從顯示記憶體中複製到主記憶體中的速度有關，複製的動作會限於 PCI Express 的速度有關，如果 block number 取得越大，要複製到 host 端的記憶體數量就越多。



這張圖是使用 0.1%的 minimum support，20 個 block number 得到的結

果，使用的 thread number 越多，就越能減少 latency 的問題。