資料科學, Fall 2017

Homework #5 CUDA B03901023 許秉鈞

---

1. Screenshot your device query results, and then paste the results in your report. (25%)

```
mkdir -p ../../bin/x86_64/linux/release
cp deviceQuery ../../bin/x86_64/linux/release
yen@doublebite-System-Product-Name:~/adrian/samples/1_Utilities/deviceQuery$ ls
deviceQuery  deviceQuery.cpp  deviceQuery.o  Makefile  NsightEclipse.xml  readme.txt
yen@doublebite-System-Product-Name:~/adrian/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 950"
  CUDA Driver Version / Runtime Version          9.0 / 8.0
  CUDA Capability Major/Minor version number:    5.2
  Total amount of global memory:                 1997 MBytes (2093547520 bytes)
  ( 6) Multiprocessors, (128) CUDA Cores/MP:     768 CUDA Cores
  GPU Max Clock rate:                            1253 MHz (1.25 GHz)
  Memory Clock rate:                             3305 Mhz
  Memory Bus Width:                              128-bit
  L2 Cache Size:                                 1048576 bytes
  Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 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 = 9.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GeForce GTX 950
Result = PASS
yen@doublebite-System-Product-Name:~/adrian/samples/1_Utilities/deviceQuery$
```

我使用的是助教提供的工作站（114.112.42.42 -p 1322）
GPU 是使用 **GeForce GTX 950** 。

2. Your code should be able to build with the provided makefile. (10%)

請直接用提供的 makefile 輸入 make 再用 ./fim.out retail.txt 0.001 outfile 就可以跑。

3. Your code should produce correct output (correct itemsets and correct support value). (15%) & 4. Your GPU version should be faster than the CPU version. (15%)

support = 0.0006
CPU (161.657 s): https://gist.github.com/AdrianHsu/81a795022b3d61b9bbbced14f380cb04
GPU (**77.524 s**): https://gist.github.com/AdrianHsu/b2ef64b9b2b96246ee6d89fa5689d620

support = 0.0007
CPU (112.888 s): https://gist.github.com/AdrianHsu/84239789217cfceea122097da211583c
GPU (**55.575 s**): https://gist.github.com/AdrianHsu/924843a3d26791e550c930d5fab7053b

support = 0.0008
CPU (89.384 s): https://gist.github.com/AdrianHsu/e85f2cf714c8641a43a5b843d07ba670
GPU (**42.766 s**): https://gist.github.com/AdrianHsu/e6e21ce0d61aaf2cf944760b143a2317

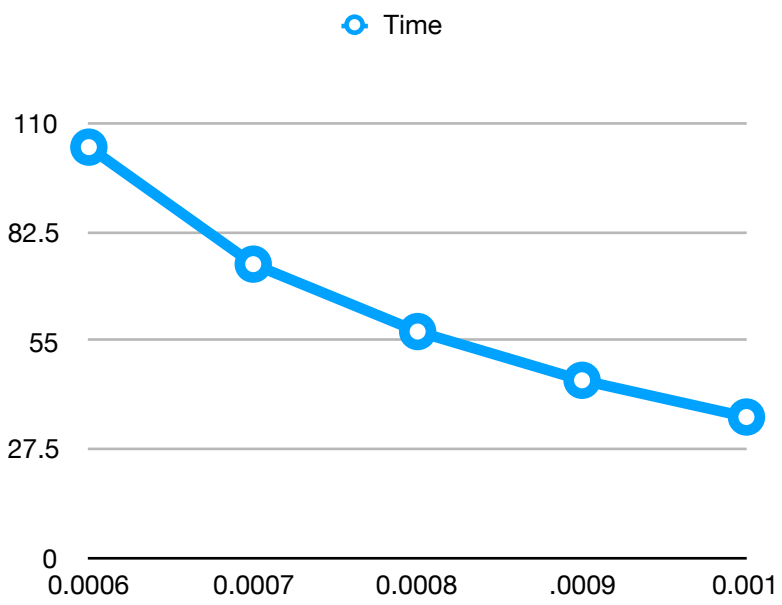
support = 0.0009
CPU (67.412 s): https://gist.github.com/AdrianHsu/dd729d603205ba792d3696028dd3db5d
GPU (**33.569 s**): https://gist.github.com/AdrianHsu/4d9410f1999b961a3ae5791a935f920a

support = 0.001
CPU (52.640 s): https://gist.github.com/AdrianHsu/eb64907a3d554154808f5fe0a2c637ef
GPU (**26.744 s**): https://gist.github.com/AdrianHsu/455d373a8f3f9ff4507947513b28881b

(參數：BLOCKNUM = 64, THREADNUM = 16)
全部都比 CPU **快了兩倍左右，如果把** THREADNUM **調到** 64 **或是** 128 **會更快。**

5. Plot the execution time of the GPU mining (5% each, 15% total)

Table 1.
控制變因：BLOCKNUM = 16, THREADNUM = 16
操縱變因：minimum Support



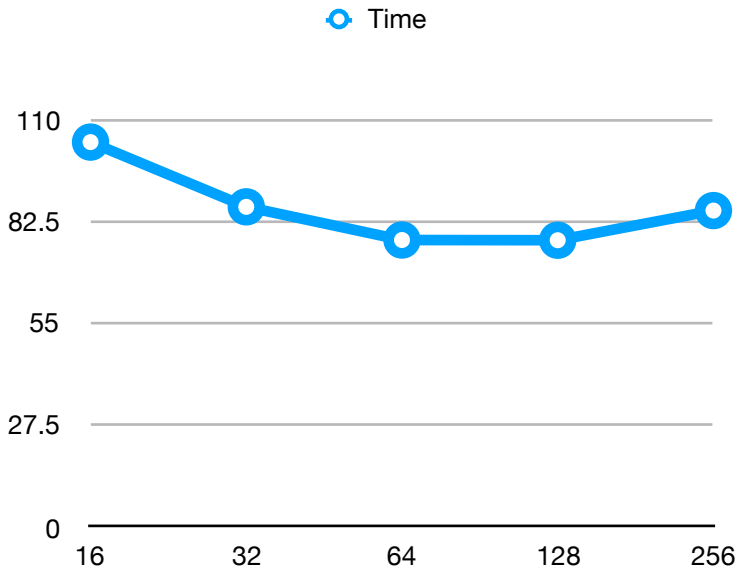
minSup	0.0006	0.0007	0.0008	0.0009	0.001
Time	103.995	74.361	57.293	44.944	**35.648**

因為 support 調的越低就會有越多 item 被包括進來，因此成長得很快，0.0006 和 0.001 跑的時間甚至是 3 倍左右。

Table 2.

控制變因：minSup = 0.0006, THREADNUM = 16

操縱變因：BLOCKNUM



BLOCKNUM	16	32	64	128	256
Time	103.995	86.491	77.524	**77.467**	85.479

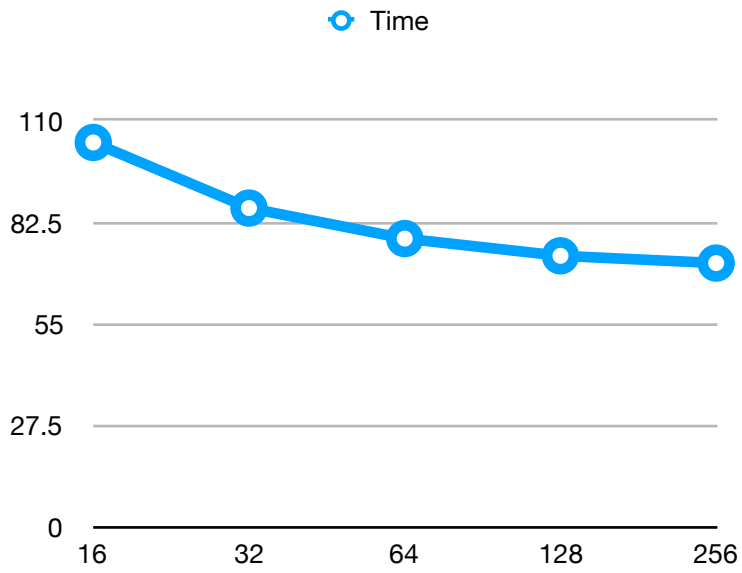
Block 越多理論上來說有更多的處理器可以平行運算，但我發現在 256 顆的時候反而慢了不少。 **The blocks in the grid are spread out over the SMs(streaming multiprocessors) to start, and then the remaining blocks are placed into a pipeline. Blocks are moved into the SMs for processing as soon as there are enough resources in that SM to take the block. In other words, as blocks complete in an SM, new ones are moved in. Therefore, having smaller blocks (128 instead of 256) may complete faster since a particularly slow block will hog fewer resources.**

reference: https://stackoverflow.com/questions/4391162/cuda-determining-threads-per-block-blocks-per-grid

Table 3.

控制變因：minSup = 0.0006, BLOCKNUM = 16

操縱變因：THREADNUM



THREADNUM	16	32	64	128	256
Time	103.995	86.265	78.018	73.359	**71.393**

第一點， 16 **比** 32 **慢這麼多是因為** warp **的單位是** 32 **條** threads**。第二點，**Thread **越多理論上來說有更多** bitwise operation **可以同時計算，原本該更快的、卻越來越平緩。我的猜測是，因為** length **不一定都會這麼大，用這麼多** thread **反而會讓** occupancy **很小（不到** 50% **之類的），再加上** shared memory **創建的時間還有** reduction **的合併時間，等等因素造成整體慢了一些些。**

reference: https://stackoverflow.com/questions/4391162/cuda-determining-threads-per-block-blocks-per-grid

6. Briefly explain your parallel design in the report. The better the design, the higher the score. (20%)

我花了很多時間在寫這次的作業（幾乎整整一個禮拜吧），其中試過了 2 - 3 **種設計但都不夠快，例如：只把最內層的** for **移出來平行運算，但結果甚至比原本慢。最後發現以下的方法有最好的結果。簡單來說，就是拆掉兩層** for loop**，把 每次的** b **用頭接著尾的方式接在一起，傳入** global function **後，在每一個** block**， 拿** a **和每一小段的** subset b **用好幾條** thread **一起算 a[k] & b[k] ， 算好存回去。就是把** for j=i+1 **這個第二層** loop **拉直成一個很長的** array **一次算好再傳回來。**

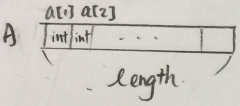
for j in (0, size):

① // 把 *a 建成 cuda array

a = eClass → items[i].db;

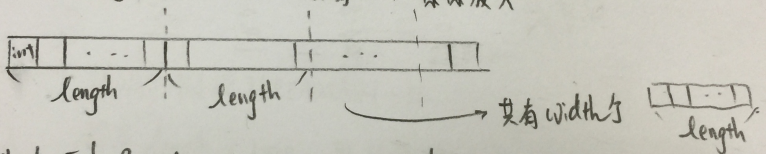
用 gpuA 存 a, 用 cudaMemcpy

gpuA



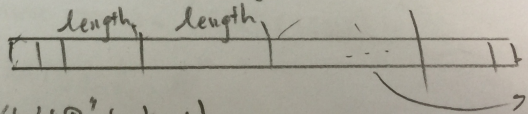
② 用 gpuB 把 for j in (i+1, size) 的所有 *b 的 db, 一條條放入

gpuB.

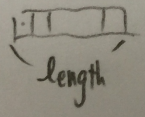


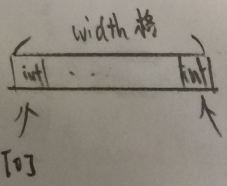
共 width 为原本 for loop 的次数, 此变成有幾條 arrays concat 在一起

③ 開 gpuTemp 和 gpuB 一樣大小,



④ 開 gpuSup 存 "原本 for loop 每次的結果" (一个 int)

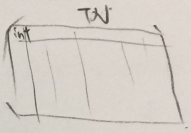
現在則是一條  經過 bitwise operation 的結果.



⑤ 進入 global eclat(), 傳入 gpuA, gpuB, gpuTemp, gpuSup 和 length, width

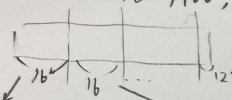
5-1 開 2-D array shared[BLOCKNUM][THREADNUM]

用 static 的, 不用 dynamic 的

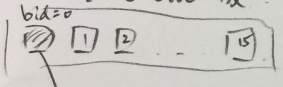


5-2. 假設 width = 1900, 而我的 BLOCKNUM = 16

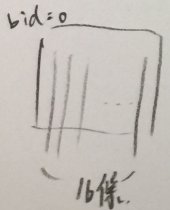
需做 1900 ÷ 16 = 118 ... 12个



分給 16 个 block 做 ↓ 下一輪



5-3 (inside one block) 假設 THREADNUM = 16.

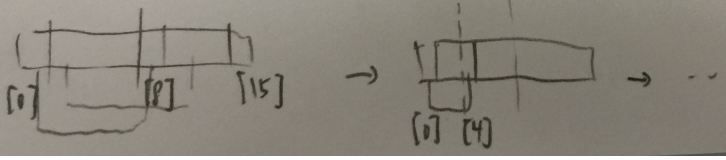


假設 length = 1000, 用 16條 threads.

把 for loop 的 bitwise operation (a[k] & b[k])

用 shared[bid][tid] 存起來.

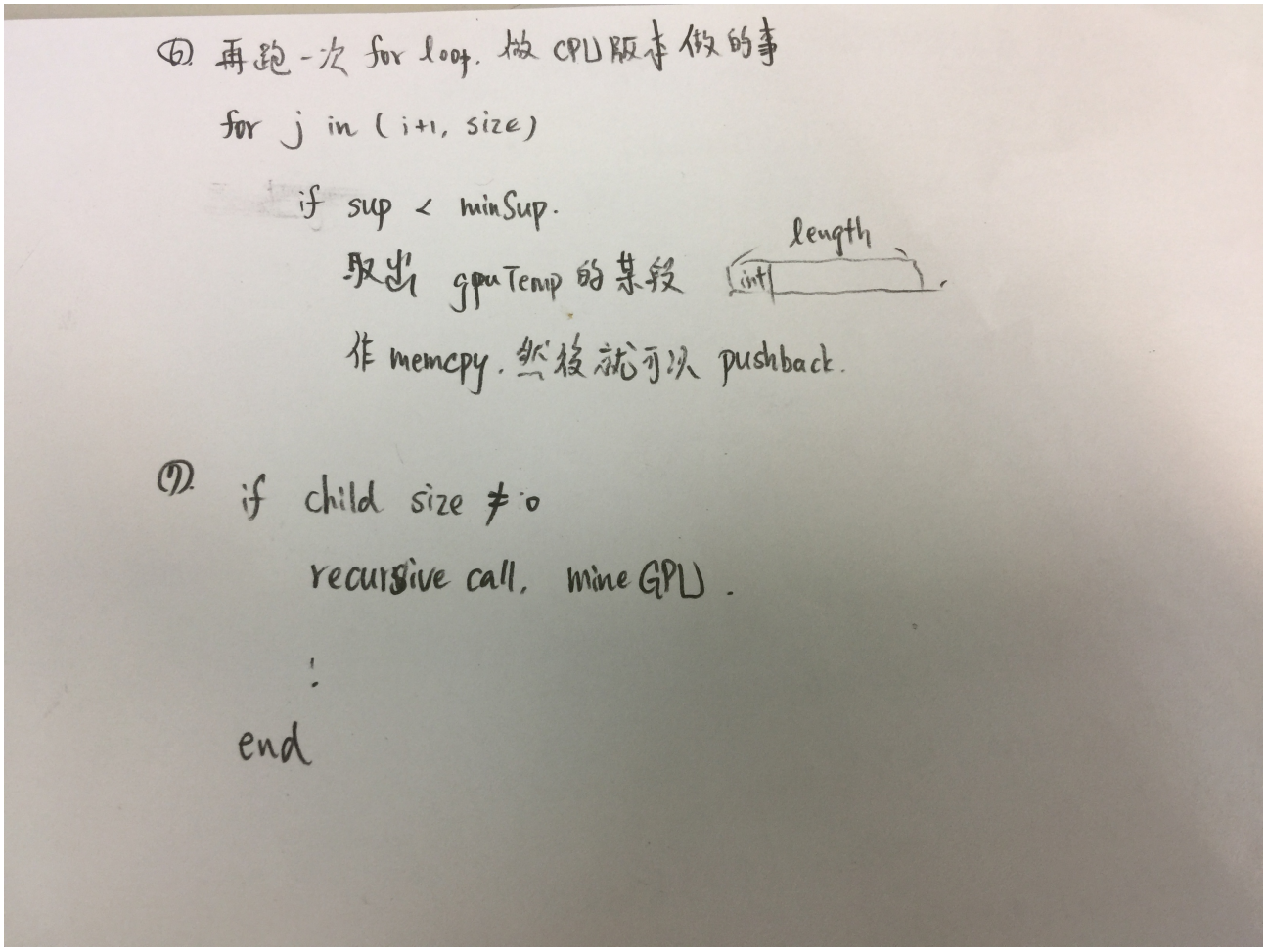
5-4 reduction technique



5-5

存回 support[width],

取 shared[bid][0] 即可



總結一下：這個平行化的程式會被以許多個 thread 來執行，數個 thread 會被群組成一個 block，而多個 thread block 則會再構成 grid。也就是一個 kernel 程式會有一個 grid，grid 底下有數個 block，每個 block 都是一群 thread 的群組。而在同一個 block 中的 thread 可以透過 shared memory 來溝通，也可以同步。最後的速度蠻好的，可以到 CPU 的兩倍速，甚至 2.5 倍，但我原本預期可以到更快，可能是因為記憶體複製比較慢、或是其他用到 CPU 的地方導致。**我覺得這次的作業是我收穫最多的一次，從完全不會 CUDA 到能夠熟練地操作 thread, block 還有 CUDA 的許多函式、同步的原理...等。**