knn CUDA program design document

袁帥

1 Used resources

我用到兩個 kernel function, 其中 computeDist 函式計算所有兩個點之間的距離平方值、並存入矩陣 D 中,knn 函式根據 D 去計算每個點的 k 個最小的距離平方值並找出對應的點.

computeDist 函式主要用到以下 resources:

- $m \times m$ 個 2-D blocks, block(i, j) 分別負責計算點 i 和點 j 之間的距離平方值.
- 每個 block 用到 n 個 1-D threads, block(i, j) 中的 thread(k) 負責計算點 i 和點 j 在第 k 維度的距離平方值. 由於 $n \in [0, 1000]$, 所以這麼設計不會超過 max number of threads/block.
- 每個 block 用到大小爲 $n \times sizeof(int)$ 的 shared memory, block(i, j) 的 shared memory 存放點 i 和點 j 在 n 個維度上的距離平方值. 由於我用到 parallel reduction 來對這些值求和, 需要經常 access, 所以把它們放到 shared memory 中. 同樣由於 n 的取值較小, 不會超出每個 block 的 shared memory size.

knn 函式主要用到以下 resources:

- m 個 1-D blocks, block(i) 分別負責求出點 i 的 k nearest neighbors.
- 每個 block 用到 m 個 1-D threads, block(i) 中的 thread(j) 負責把點 i 到其他點的距離平方值從 global memory 中 load 進 shared memory. 由於 $m \in [10, 1000]$, 所以這麼設計不會超過 max number of threads/block.
- 每個 block 用到大小爲 $m \times sizeof(int)$ 的 shared memory, block(i) 的 shared memory 存放點 i 到其他點的距離平方值. 這是因爲對每個點來說, 求 k nearest neighbors 只需要這 m 個值, 而且這些值經常要被 access, 所以我把它們放到 shared memory 中. 同樣由於 n 的取值較小, 不會超出每個 block 的 shared memory size.

2 Details of implementation

1. computeDist 求兩點之間的距離平方值. 由於 $D_{i,j} = D_{j,i}$, 而且我們在本問題不考慮點到自身的情況, 所以只需對 $i > j$ 的情況進行運算就可以了. 滿足 $i > j$ 的 block(i, j) 裏的 n 個 thread 先分別去算 n 個 dimension 的距離平方值, 再通過 parallel reduction 的方式求和. 之後處理 n 爲奇數 parallel reduction 漏算一個元素的問題. 最後把結果 write back 到 global memory 的 D 中. 考慮寫入 global memory 速度較慢, 因此重複的 $D_{i,j}$ 和 $D_{j,i}$ 也只寫一項.

2. 呼叫 cudaDeviceSynchronize(), 保證每個 block 已經執行完 computeDist 這個 kernel function, 此時所有點與點之間的距離平方值已完全寫入到 D 中.

3. 對每個點, 把它到其他 (m-1) 個點的距離平方值從 global memory 的 D 中 load 進每個點相應 block 的 shared memory. 如上所述,D 中不存放冗餘信息, 所以當 $i < j$ 時需要將 i, j 互調來找到正確的 $D_{i,j}$ (即 $D_{j,i}$).

4. 讓每個 block 的第一個 thread 來找 k 個最小值並把相應的 index 寫到 out 中.

3 Improvement

我想到了幾種可能可以進一步改進的方法:

- 求 k 個最小值的部分可以再用一次 parallel reduction. 不過求完前 $(x-1)$ 個最小值後求第 x 個最小值時, 需要把 D 的內容再重新 load 進 shared memory 中, 這部分可能反而會降低 performance. 同時 index 需要額外的 memory 來存放, 而且由於題目要求兩個距離相等時選擇二者 index 小的那個, 所以在 parallel reduction 中也需要在距離相等時繼續對 index 進行比較. 這些問題或許有改進的 parallel reduction 可以解決.

- 存放 k nearest neighbors 的 index 的 out, 由於在 k 次 iteration 中也需要被 access 以確保前面求出的最近鄰居不會再次出現, 所以也可以先放到 shared memory 中, 再寫回到 global memory.

- 對於爲了去掉 $D_{i,j} = D_{j,i}$ 冗餘而引起的 $i > j$ 的 block(i ,j) 閒置的 skew workload 問題, 可以通過只分配 $m(m-1)/2$ 而非 $m \times m$ 個 block 來解決, 這需要重新讓 block(i, j) 對應到正確的兩個點上 (此時已不一定是對應到點 i 和點 j!), 不過對於程式執行時間並沒有提升.