Chiricu Miruna
332CB

Tema 3 – ASC

Ideea de implementare pentru HashTable:
- vector de perechi: cheie – valoare

Adaugarea unui element:
- functie pentru GPU, care pentru fiecare pereche cheie-valoare din listele date de insertBatch:

* Daca cheia si valoarea sunt valide, continua la pasul urmator

* Face hash-ul prin Multiplication Method: `hash = capacity * (fmod(key * A, 1.0));`
  unde A e o constanta egala cu $0.5 \times (\sqrt{5} - 1)$

* Rezolva coliziuni (daca exista):
  - atomicCAS(adresa, val_comp, new_val) este folosit pentru ca in fiecare thread incearca elementul corespunzator hash-ului, apoi le incearca liniar la dreapta pana la sfarsitul array-ului si o ia de la capat, sa vada daca sunt zone libere (sau chiar gaseste cheia deja pusa la un batch anterior) unde ar putea pune elementul. Fiecare thread va face asta, deci operatia de verificare si modificare a cheii trebuie sa fie atomica si serializabila. E clar ca in caz ca exista deja cheia in hash_table, ori a fost pusa pe pozitia corecta, ori s-au incercat pozitiile liniar si nu va exista un 0 inainte de aceasta cheie => E safe sa incercam prima oara daca exista cheia, apoi daca exista 0 (slot liber).

* in caz ca este pus un element nou (pe o pozitie libera, nu un update), se updateaza variabila existenta si pe Host si pe GPU: slots_occ

Extragerea unui element:
- face exact ce se face pentru a adauga un element, doar ca nu modifica nimic

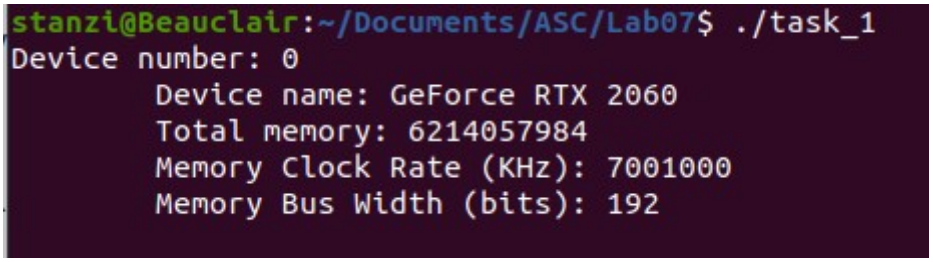
Copierea cheilor si a valorilor la redimensionare:
- copiem din hash_table intr-un batch de chei si de valori (folosit la redimensionare)

reshape:
- aloca un batch de chei si de valori de size = capacitatea hash_table-ului
- le copiaza printr-un kernel la device
- dezaloca memoria pentru hash_table
- reinitializeaza capacity si tot hash_table-ul cu noua capacitate
- apeleaza insertBatch cu batch-ul de la pasii anteriori

insertBatch:
- explicatie pentru felul in care se calculeaza noua posibila capacitate in comentarii
- verificam daca trebuie sa facem reshape => apelam reshape daca e necesar
- alocam memoria, adaugam elementele, dezalocam memoria

getBatch:
- aloca memoria, apeleaza getBatch, dezaloca memoria

PS: Pe local imi trec toate testele, dar placa mea grafica e de doua ori mai slaba decat cea de pe coada.



Pe coada nu trec ultimele 3 teste si nu am putut sa o scot din **(test_map.cpp, 256): minSpeed > avgSpeedInsert: File exists**

Am viteze de doua ori mai proaste pe placa mea, deci de aceea trec. Dar ce e cu minSpeed? De ce nu ar trebui sa fie sub o viteza minima? Nu ar trebui sa fie de bine?

Am jonglat cat am putut cu load_factor si cu resize_factor. Nu a functionat.

P.S: Am avut noroc ca am nVidia si instalasem cuda pe local (dual boot, ca pe wsl a fost criminal, si nu am reusit dupa 2-3h). Pe coada dura mai mult de 15min pentru un job uneori. Totodata, am avut nenorocul sa nu treaca toate testele pe coada.

```
TERMINAL  DEBUG CONSOLE  PROBLEMS  OUTPUT                    2: bash

------- Test T1 START   ----------

b'HASH_BATCH_INSERT   count: 1000000          speed: 98M/sec          loadfactor: 60%
\nHASH_BATCH_GET      count: 1000000          speed: 550M/sec         loadfactor: 60%
\n-----------------------------------------\nAVG_INSERT: 98 M/sec,   AVG_GET: 550 M/sec,
    MIN_SPEED_REQ: 10 M/sec \n\n'
------- Test T1 END     ----------       [ OK RESULT: +20 pts ]



------- Test T2 START   ----------

b'HASH_BATCH_INSERT   count: 500000           speed: 89M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 500000           speed: 69M/sec          loadfactor: 60%
\nHASH_BATCH_GET      count: 500000           speed: 450M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 500000           speed: 426M/sec         loadfactor: 60%
\n-----------------------------------------\nAVG_INSERT: 79 M/sec,   AVG_GET: 438 M/sec,
    MIN_SPEED_REQ: 20 M/sec \n\n'
------- Test T2 END     ----------       [ OK RESULT: +20 pts ]



------- Test T3 START   ----------

b'HASH_BATCH_INSERT   count: 125000           speed: 66M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 125000           speed: 48M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 125000           speed: 32M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 125000           speed: 135M/sec         loadfactor: 80%
\nHASH_BATCH_INSERT   count: 125000           speed: 26M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 125000           speed: 137M/sec         loadfactor: 72%
\nHASH_BATCH_INSERT   count: 125000           speed: 95M/sec          loadfactor: 84%
\nHASH_BATCH_INSERT   count: 125000           speed: 18M/sec          loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 301M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 296M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 292M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 297M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 298M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 295M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 296M/sec         loadfactor: 60%
\nHASH_BATCH_GET      count: 125000           speed: 210M/sec         loadfactor: 60%
\n-----------------------------------------\nAVG_INSERT: 70 M/sec,   AVG_GET: 286 M/sec,
    MIN_SPEED_REQ: 40 M/sec \n\n'
------- Test T3 END     ----------       [ OK RESULT: +15 pts ]



------- Test T4 START   ----------

b'HASH_BATCH_INSERT   count: 2500000          speed: 97M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 2500000          speed: 72M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 2500000          speed: 41M/sec          loadfactor: 60%
\nHASH_BATCH_INSERT   count: 2500000          speed: 169M/sec         loadfactor: 80%
\nHASH_BATCH_GET      count: 2500000          speed: 531M/sec         loadfactor: 80%
\nHASH_BATCH_GET      count: 2500000          speed: 503M/sec         loadfactor: 80%
\nHASH_BATCH_GET      count: 2500000          speed: 286M/sec         loadfactor: 80%
\nHASH_BATCH_GET      count: 2500000          speed: 259M/sec         loadfactor: 80%
\n-----------------------------------------\nAVG_INSERT: 95 M/sec,   AVG_GET: 395 M/sec,
    MIN_SPEED_REQ: 50 M/sec \n\n'
------- Test T4 END     ----------       [ OK RESULT: +15 pts ]



------- Test T5 START   ----------

b'HASH_BATCH_INSERT   count: 20000000         speed: 107M/sec         loadfactor: 59%
\nHASH_BATCH_INSERT   count: 20000000         speed: 74M/sec          loadfactor: 59%
\nHASH_BATCH_GET      count: 20000000         speed: 346M/sec         loadfactor: 59%
\nHASH_BATCH_GET      count: 20000000         speed: 331M/sec         loadfactor: 59%
\n-----------------------------------------\nAVG_INSERT: 90 M/sec,   AVG_GET: 338 M/sec,
    MIN_SPEED_REQ: 50 M/sec \n\n'
------- Test T5 END     ----------       [ OK RESULT: +15 pts ]

TOTAL gpu_hashtable  85/85
stanzi@Beauclair:~/Documents/ASC/Tema3$
```