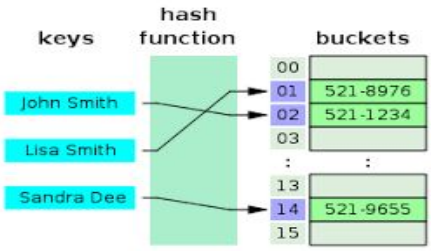
Parallel hashtable



Last update: 18 Aprilie 2019
Publicare: 22 Aprilie 2019
Soft deadline: **6 mai 2019, ora 23:55**
Hard deadline: **13 mai 2019, ora 23:55**

Se va implementa o structura de date tip hashtable folosind CUDA avand ca target la alegere GPU Tesla K40 (hp-sl.q) sau GPU Tesla 2050/2070 (ibm-dp.q).

Aceasta structura de date tip hashtable va avea urmatoarele caracteristici:

- va face maparea key -> value, (o singura valoare per key)
- va putea stoca date de tip int32, mai mari strict ca 0, atat pentru chei cat si valori
  - valid (key->value): 1234 -> 8965, 12334244 -> 8223965
  - invalid (key->value): 0 -> 9779, -12334244 -> 83965, 7868 -> -977, “jknjjk” -> 78
- va stoca datele in VRAM si va putea sa isi reajusteze dimensiunea cat sa poata acomoda numarul de perechi key->value, avand un loadFactor decent (sloturi ocupate / sloturi disponibile)
- va rezolva intern coliziunile (ex prin resize, folosind multe functii de hash etc)
  - functiile hash vor fi pentru cazurile generale (ex. cele de tip ax % b, a si b prime)
  - exemplu de functie hash neaceptata: h(x) = x % limit, atunci cand cheile sunt tot timpul in ordine crescatoare pe intervale si unice (nu trateaza cazul general)
- va face update la valoarea unei chei -> operatia de insert nu va contine niciodata 2 chei identice, insa operatii succesive de insert nu au restrictii (se va face update de valoare)
- va putea intoarce corect, intr-un timp rapid, valoarea corespunzatoare unei chei
- va intoare incarcarea (memoria efectiv folosita / memoria alocata) via functia load_factor
  - o implementare gresita sau neclara poate duce la o depunctare suplimentara

Implementarea temei are design la liber cat timp testele trec si solutia este una rezonabila, neavand ca scop trecerea testelor (ex. <<asa nu>> comportamentul tabelei hash e strict modelat pe teste, cand se face resize etc).

Exemple de implementari:

- **Linear probing** - se calculeaza hash(key) si se daca slotul nu e liber se incearca succesiv hash(key) + 1… hash(key) + N
- **Cuckoo based** - se folosesc 2-3 functii hash si se cauta un slot liber, alternativ se face evict la key2-value2 din acel slot, se insereaza noua pereche key-value iar key2-value2 se cauta sa se insereze folosind o alta functie hash din cele disponibile
- **Bucketized cuckoo** - se foloseste tehnica cuckoo avand buckets cu 2,4,8 sloturi fiecare.

Schelet tema

Se va completa scheletul pus la dispozitie (gpu_map.cu) avand urmatoarele functii ce trebuie implementate:

```
/* INIT HASH
 */
GpuHashTable::GpuHashTable(int size) { }

/* DESTROY HASH
 */
GpuHashTable::~GpuHashTable() { }

/* RESHAPE HASH
 */
void GpuHashTable::reshape(int numBucketsReshape) { }

/* INSERT BATCH
 */
bool GpuHashTable::insertBatch(int *keys, int* values, int numKeys) { return false; }

/* GET BATCH
 */
int* GpuHashTable::getBatch(int* keys, int numKeys) { return NULL; }

/* GET LOAD FACTOR
 * num elements / hash total slots elements
 */
float GpuHashTable::loadFactor() {
	return 0.f; // no larger than 1.0f = 100%
}

#define HASH_INIT GpuHashTable GpuHashTable(1);
#define HASH_RESERVE(size) GpuHashTable.reshape(size);

#define HASH_BATCH_INSERT(keys, values, numKeys) GpuHashTable.insertBatch(keys,
values, numKeys)
#define HASH_BATCH_GET(keys, numKeys) GpuHashTable.getBatch(keys, numKeys)

#define HASH_LOAD_FACTOR GpuHashTable.loadFactor()
```

Notare

Se pot adauga alte fisiere care sa fie incluse in program insa nu se vor altera fisierele puse la dispozitie, exceptie fiind **Makefile, gpu_hashtable.cu** si **gpu_hashtable.hpp**.

Punctajul maxim este de 100 pct distribuite astfel:

10 pct	Implementare descrisa in README, alaturi de rezultate si o discutie asupra lor Programul compileaza, codul nu are disfunctionalitati majore
90 pct	Punctaj dat de bench.py, performanta trebuie sa fie similara si in alte configuratii A nu se optimiza solutia pentru trecerea testelor !

Arhiva va cuprinde obligatoriu:

- Fisierele cu solutia si alte fisiere aditionale folosite, dar minim:
  - gpu_hashtable.cu
  - gpu_hashtable.hpp
- Makefile, daca sunt necesare modificari
- Readme, unde se explica:
  - Cum s-a implementat solutia ?
  - Cum se stocheaza hashtable in memoria GPU VRAM ?
  - Output la performantele obtinute si pe ce cluster s-a rulat.
  - Discutie si/sau analize suplimentare (ex nvprof).