Tema 3 - GPU Hashtable

- **Soft + hard deadline:** 31 Mai 2021
- **Responsabil:** Grigore Lupescu, Razvan Dobre

- Dată publicare: 14 Mai 2021
- Dată actualizare: 14 Mai 2021

Enunț

Se va implementa o structură de date tip hashtable folosind CUDA, având ca target GPU Tesla K40 (hp-sl.q).

Tema se poate dezvolta pe orice sistem cu suport CUDA, dar punctarea se face exclusiv pe coada hp-sl.q

Această structură de date tip hashtable va avea următoarele caracteristici:

1. va face maparea key → value, (o singură valoare per key)
2. va putea stoca date de tip int32, mai mari strict ca 0, atât pentru chei, cât și pentru valori
   - valid (key→value): 1234 → 8965, 12334244 → 8223965
   - invalid (key→value): 0 → 9779, -12334244 → 83965, 7868 → -977, “jknjjk” → 78
3. va stoca datele în VRAM și va putea să își reajusteze dimensiunea cât să poată acomoda numărul de perechi key→value, având un loadFactor decent (sloturi ocupate / sloturi disponibile)
4. va rezolva intern coliziunile (exemplu: prin resize, folosind multe funcții de hash etc.). Funcțiile hash vor fi pentru cazurile generale (ex. cele de tip ax % b, a si b prime). Exemplu de funcție hash neacceptată: h(x) = x % limit, atunci când cheile sunt tot timpul în ordine crescătoare pe intervale și unice (nu tratează cazul general)
5. va face update la valoarea unei chei → operația de insert nu va conține niciodată 2 chei identice, însă operații succesive de insert nu au restricții (se va face update de valoare)
6. va putea întoarce corect, într-un timp rapid, valoarea corespunzătoare unei chei
7. va întoarce încărcarea (memoria efectiv folosită / memoria alocată) via funcția load_factor
8. implementare greșită sau neclară poate duce la o depunctare suplimentară

Implementarea temei are design la liber, cât timp testele trec și soluția este una rezonabilă, neavând ca scop trecerea testelor (ex. «așa nu» comportamentul tabelei hash e strict modelat pe teste, când

se face resize etc).

Schelet tema

Se va completa scheletul pus la dispoziție (gpu_map.cu) având următoarele funcții 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; }
```

Alocarea și dealocarea memoriei CUDA se va face folosind doar urmatoarele functii:

```
cudaError_t GpuAllocator::_cudaMalloc( void** devPtr, size_t size );
cudaError_t GpuAllocator::_cudaMallocManaged( void** devPtr, size_t size );
cudaError_t GpuAllocator::_cudaFree( void* devPtr );
```

Acestea sunt wrappers pentru funcțiile originale CUDA, însă verifică dimensiunea memoriei alocate, cât să nu se depășească semnificativ scopul problemei. Ele sunt folosite, de asemenea, în calculul memoriei efectiv folosite, pentru a determina hashtable load factor.

Notare

Se pot adăuga alte fișiere care să fie incluse în program, însă nu se vor altera fișierele puse la dispoziție, exceptie fiind Makefile, gpu_hashtable.cu si gpu_hashtable.hpp.

Punctajul maxim este de 100 pct distribuite astfel:

- [**85 pct**] Punctaj dat de bench.py, performanța trebuie să fie similară și în alte configurații. A nu se optimiza soluția pentru trecerea testelor!
- [**15 pct**] Implementare descrisă în README, alături de rezultate și o discuție asupra lor. Programul compilează, codul nu are disfuncționalități majore

Arhiva zip va cuprinde obligatoriu fișierele cu soluția și alte fișiere adiționale folosite, dar minim:

- gpu_hashtable.cu
- gpu_hashtable.hpp
- Readme, unde se explică:
  - Cum s-a implementat soluția ?
  - Cum se stochează hashtable în memoria GPU VRAM ?
  - Output la performanțele obținute și discutie rezultate.

Precizări și recomandări

În cazul în care job-urile vă rămân “agățate”, va recomandam să utilizați de pe fep.grid.pub.ro, comanda

```
qstat
```

pentru a vedea câte job-uri aveți pornite, și apoi să utilizați comanda

```
qdel -f <id-sesiune>
```

unde <id-sesiune> sunt primele cifre din stânga, rezultate după comanda **qstat**.

Sesiunile interactive deschise prin qlogin nu sunt permise pe coada hp-sl.q. Va trebui să utilizați qsub pentru a folosi această coadă.

Resurse

- Cluster cheat sheet
- Document Tema3 CUDA Hashtable
- Schelet de cod