Adatpárhuzamos elérést biztosító láncolt lista kifejlesztése és implementációja GPU-n

ÓE NIK

Szoftvertervezés és -fejlesztés specializáció

Varga Tamás Gábor

Konzulens: Szénási Sándor

Absztrakt

A jelenlegi folytonos területkiosztású adatszerkezet implementációk alkalmatlanok a komplex szétszóró beillesztésekre, a nem folytonos területkiosztású adatszerkezet implementációk pedig csak éppen elfogadható hatékonysággal képesek ugyanerre a műveletre. Ezért célszerű kidolgozni egy új adatszerkezetet az alábbiak szerint: hatékony komplex beillesztés és törlés műveletek megvalósítása, memóriaigény minél kisebbre való redukálása, hatékony reallokáció megvalósítása, CPU és GPU közötti függőségek csökkentése. Az elkészült GPU alapú lista adatszerkezet a vizsgálatok alapján több nagyságrenddel gyorsabb, mint a napi gyakorlatban használt, programozási nyelvekbe épített alapvető adatszerkezetek.

Kulcsszavak

Adatpárhuzamos programozás, Adatszerkezetek, Láncolt lista, GPU programozás, Program optimalizálás

Motiváció

Számos láncolt adatszerkezet implementációt használnak a napi gyakorlatban, a GPU-k megjelenésével azonban célszerű megvizsgálni, hogy ez az adatpárhuzamos eszköz tud-e újat mutatni ezen a területen is. Az elemi adattárolás területén nem várható jelentős előrelépés, a komplex egyszerre több elemmel dolgozó műveletek esetén azonban jelentős lehet az architektúra előnye. A CUDA fejlesztői környezeten belül eddig nem létezett olyan komplex dinamikus adatszerkezet, amellyel a CPU és a GPU közötti függőségeket nagymértékben redukálni lehetne, ezért érdemes ebbe az irányba megtenni a szükséges lépéseket.

Hasonló fejlesztések

Számos láncolt lista megvalósítás készült az évtizedek folyamán, amelyek mindenki számára elérhetőek. Ezek közül az alábbiak értékelése történt meg:

- CPU alapú megvalósítások: ArrayList, Vector, List adatszerkezetek. Ezek működésüket tekintve meglehetősen hasonlók.
- GPU alapú megvalósítás: CUDA könyvtárai között található thrust csomag vector adatszerkezete

Az adatszerkezetek vizsgálata alapján megállapítható, hogy azok nem tudják hatékonyan kezelni a komplex beszúrás és törlés műveleteket. A komplex szétszóró beillesztés alatt azt a műveletet értjük, amikor az adatszerkezetnek nem csak egy pontján szeretnénk új elemeket beszúrni, hanem egy megadott feltétel (predikátum) alapján egyszerre akár több helyre is. A beillesztés konstans és variáns elemszámú is lehet, ami azért fontos, mert ez utóbbi esetben a megvalósítás jóval összetettebb, illetve a memória intenzitása is nagyobb.

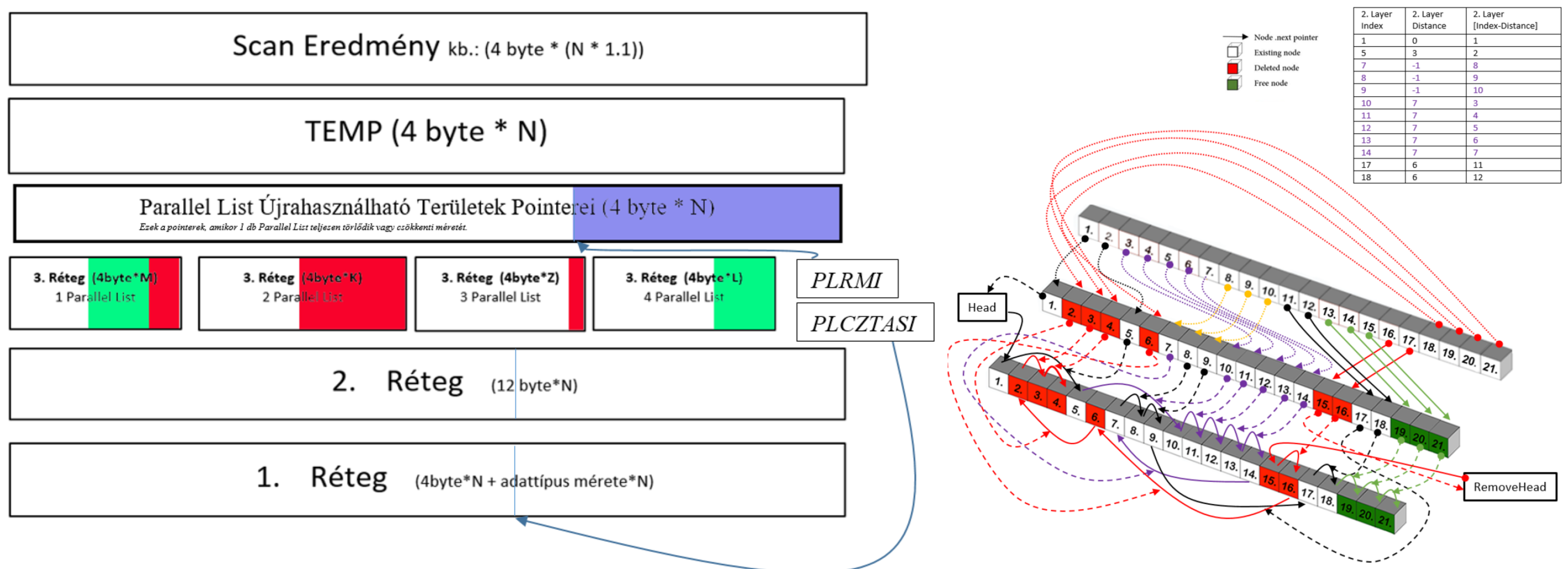
Sikerek

- Kari TDK 1. helyezett
- OTDK különdíj
- 1 konferenciacikk

Saját módszer bemutatása

Az új adatszerkezet kifejlesztése CUDA C-re történt. A fejlesztés szempontjából nem az volt az elsődleges cél, hogy minden masszívan párhuzamos architektúrán elérhető legyen a megoldás, hanem első lépésként csak az, hogy a megtervezett adatszerkezetet és annak műveleteit meg lehet-e egyáltalán valósítani.

Az implementáció a törölt elemek területeit felcsatolja a lista végeire, ezáltal ezek a területek újra felhasználhatóvá válnak. Az ábrán látható piros színű területek a törölt elemek területei, a zöldek a szabad területek. A PLRMI egy Parallel LIFO Buffer-re mutat. Ezzel a speciális kialakítással lehetővé vált, hogy a kiválasztott listába a zöld vagy piros helyekre is lehetőség legyen az adatokat felvinni. Amennyiben ezek a területek telítődnek, akkor PLRMI területekre kerülnek fel az adatok, ha pedig azok a területek is telítődtek, és még mindig vannak felírandó adatok, akkor a PLCZTASI (Parallel List Current Zero Tempory Address Iterator) helyére ugrik az implementáció és oda írja fel az adatokat.



Az adatszerkezet az alábbi rétegekből áll:

- Az alsó réteg 4 byte .next mutató, emellé társul a konkrét adat, amelynek méretét az adat típusa határozza meg. T jelöli az eltárolandó adattípus méretét (1, 4, 8, stb. byte).
- A második réteg 4 byte mutatóval rendelkezik, ami az első réteg .next mezőire mutat, illetve a head elemre. Ezeken kívül rendelkezik egy távolság szám értékkel, és 2 bool változóval, amik 1 byte méretűek, de ha egyetlen struct-ban adjuk meg, akkor a két logikai érték egy 4 byteos blokkban helyezkedik el.
- A harmadik réteg az olvasási réteg, amely egy 4 byte méretű mutatót tartalmaz, ami a 2. réteg hozzátartozó memóriacímére mutat.

Az első adatszerkezet a műveletek kutatására, fejlesztésére és tesztelésére lett kialakítva. Több ezt követő lépésben sikerült az algoritmust optimalizálni, ezzel csökkenteni a futásidőt és a memóriaigényt.

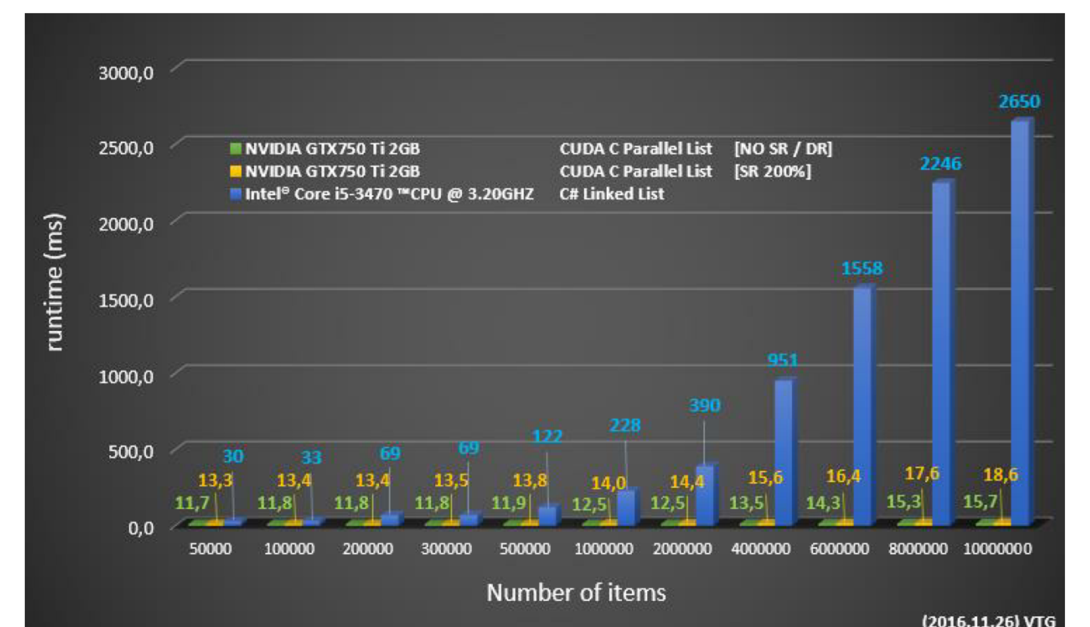
Eredmények értékelése

A fejlesztés sikeres volt, nagyjából sikerült megközelíteni a hagyományos adatszerkezetek memóriaigényét nagyságrendekkel jobb sebesség mellett: C# LinkedList x86 (20 byte + T), x64 (48 byte). Az új fejlesztésű Parallel List x86 (13 byte + T), x64 (25 byte + T). Ahol T az érték típusa szerint változó 1, 4, 8 byte.

Az eredmények értékelése egyszerű, hiszen jól számszerűsíthetők a különféle futásidők és szükséges memóriamennyiségek. Számos teszt lett elvégezve különböző műveletekkel és elemszámokkal.

A folytonos területkiosztású adatszerkezetek összehasonlításánál csak az 50000 elemszámú komplex szétszóró beillesztés lett pontosan megmérve. A C# List 2 perc, a Cuda C Thrust Vektor 4,5 perc, a kifejlesztett Parallel List pedig 12 milliszekundum alatt végezte el a komplex szétszóró beillesztést. A további elemszámú komplex beillesztések azért nem kerültek mérésre, mert például 10 millió elem esetén a mindegyik eleme mögé új adatok szétszórása több óráig is eltarthat.

A következő teszt eredmények mutatják a Parallel List összehasonlítását a nem folytonos területkiosztású adatszerkezet implementáció eredményeivel. A teszteredmények a komplex beszúrás időigényét mutatják. Látható, hogy az elemszám növekedésével jelentősen növekszik a GPU alapú megvalósítás előnye. A 10 millió elemmel rendelkező Parallel List a legrosszabb esetben is 2,4× hatékonyabb, a legjobb esetben pedig 169× a hatékonyabb, mint az általánosan használt C# LinkedList adatszerkezet.



Az egyszerű törlést a Parallel List 2,8 milliszekundum a thrust vektor 4,1 milliszekundum a C# List, pedig 7 milliszekundum alatt végezte el. A feltétel alapú törlésnél is hasonlóak az arányok.