Szimuláció RICHARD M. KARP és AVI WIGDERSON
„A Fast Parallel Algorithm for the Maximal Independent Set Problem” című cikke alapján

(Készítette: Domoszlai László)

1. Bevezetés

A következőkben megadott algoritmus EREW párhuzamos gépen $O((\log n)^4)$ idő alatt oldja meg a maximális független halmaz keresés problémáját $O(n^3/(\log n)^3)$ processzor segítségével. Az implementáció JAVA nyelven történt a jgrapht gráf könyvtár felhasználásával. Az algoritmus leírása implementációs kérdésekre nem tér ki, tehát szabad kezet kaptam az implementációra, kivéve, hogy a gráf műveletek megvalósítására nem volt befolyásom. Továbbá a tesztkörnyezet (egy kétprocesszoros 64 bites számítógép) is erősen befolyásolta az implementációt. Nagyszámú rövid futási idejű taszk erősen csökkentette a futási időt, ezért bizonyos részek párhuzamosítását el is hagytam.

2. Az algoritmusról

Az algoritmus alapötlete az, hogy a független csúcshalmazunkat $O((\log n)^2)$ iteráció alatt állítsuk elő. Az algoritmus vázlatosan a következő:

```
I ← Ø; H←V;
while H≠Ø do
begin
   S←egy független halmaz a H indukált részgráfjában;
   I←I U S;
   H←H – (S U N_H(S));
end;
```

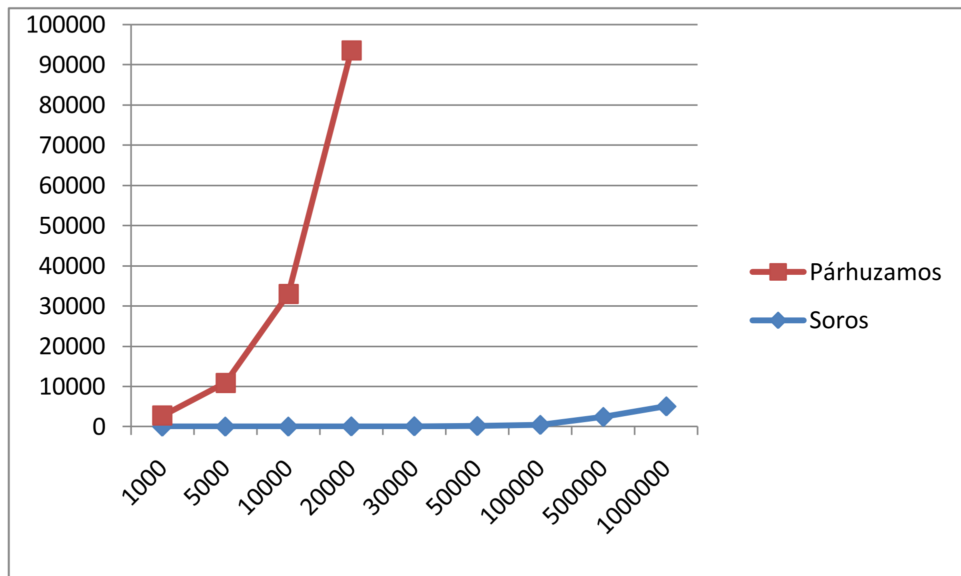
A kívánt lépésszám úgy érhető el, ha $|S \cup N_H(S)| = \Omega(|H|/\log|H|)$. Az algoritmus legnagyobb része az ilyen tulajdonságú S halmaz előállítását célozza. Ennek lényege egy pontszámítási rendszer, amellyel becsülni lehet, hogy megfelelő tulajdonságú-e a halmaz. Ez a pontszám annál nagyobb, minél nagyobb a halmazban található csúcsok fokszáma. Első lépésben tehát keresnünk kell a H egy olyan részhalmazát, amelyben a relatív magas fokszámú elemek száma elég nagy. Az így kapott halmaznak előállítjuk az összes, megadott paraméternek megfelelő BDDI (Balanced Incomplete Block Design) részhalmazát, és ezek közül kiválasztjuk a legnagyobb pontszámmal rendelkezőt. Végül az így kapott halmazt függetlenné tesszük, a benne szereplő élek tetszőleges csúcsainak eliminálásával. Az algoritmus vázlata ezek után:

```
I ← Ø; H←V;
while H≠Ø do
begin
   K←HEAVYFIND(H);
   T←SCOREFIND(H);
   S←INDFIND(T);
   I←I U S;
   H←H – (S U N_H(S));
end;
```

Az így kapott algoritmusban aztán az egyes lépések külön-külön jól párhuzamosíthatóak, de a felhasznált munka becsléséhez érdemes az algoritmust egyelőre párhuzamosítás nélkül vizsgálni.

3. Az algoritmus elemzése párhuzamosítás nélkül

Látható, hogy az új algoritmusunk elég bonyolult, várhatóan sokkal több munkát végez, mint az eredeti soros algoritmus. Ezt a mérések is megerősítik:

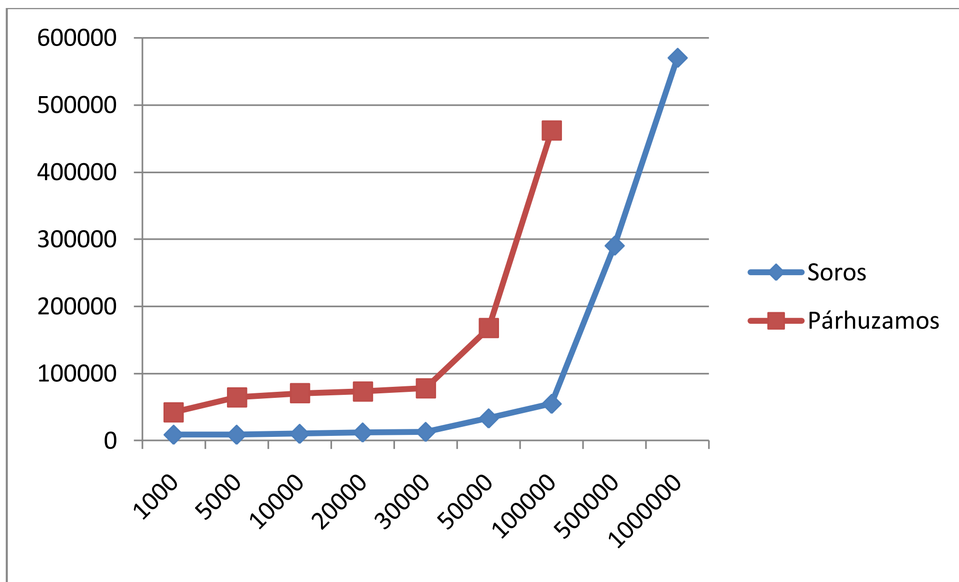


1. ábra

A függőleges tengelyen a futási idő látható milliszekundumban, a vízszintes tengelyen pedig a gráf csúcsszáma. A párhuzamos algoritmusnál csak 20 000 csúcsig végeztem el a méréseket, így is látható az óriási eltérés a futási időben.

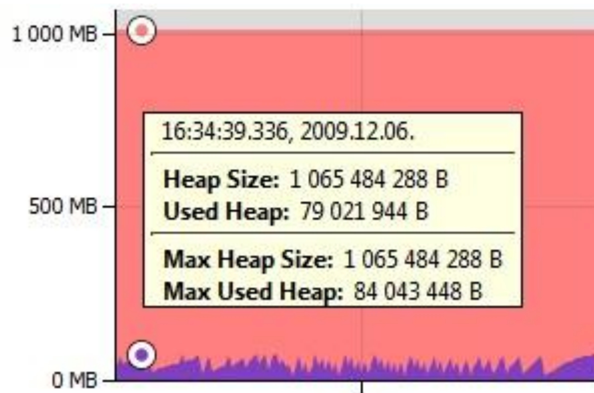
Most vizsgáljuk meg az algoritmusok memória felhasználását. A 2. ábrán látható diagram függőleges tengelye a maximális memória felhasználás kilóbájtban, a vízszintes pedig a gráf

csúcsszáma. Megfigyelhető, hogy bár a két algoritmus memória felhasználási görbéje hasonló, a soros csúcsszámhoz viszonyított memória felhasználása mégis sokkal nagyobb (lesz), mint a párhuzamosé, mivel a párhuzamos algoritmus memória felhasználásában ugrások tapasztalhatók (ez itt sajnos nem látszik jól, mert 50 000 csúcs után nem elég sűrű a mintavétel a párhuzamos algoritmus futási ideje miatt, de 5 000 és 30 000 csúcs között például a memória felhasználás gyakorlatilag nem változik) és köztük a memória felhasználás nem nő lényegesen, míg a soros algoritmusnál folyamatos növekedés tapasztalható. Ez azért van, mert a SCOREFIND eljárásban az aktuális csúcshalmazunkhoz kapcsolódó 2 hatványnál ($\max\{2^{l'}-1 \mid l'$ egész és $2^{l'}-1 \in [1,(|H|/\log|H|)]\}$) kisebb egyenlő kitevőjű 2 hatvány méretű részhalmazokat állítunk elő és ezek a részhalmazok határozzák meg nagyságrendileg az algoritmus memória felhasználását.



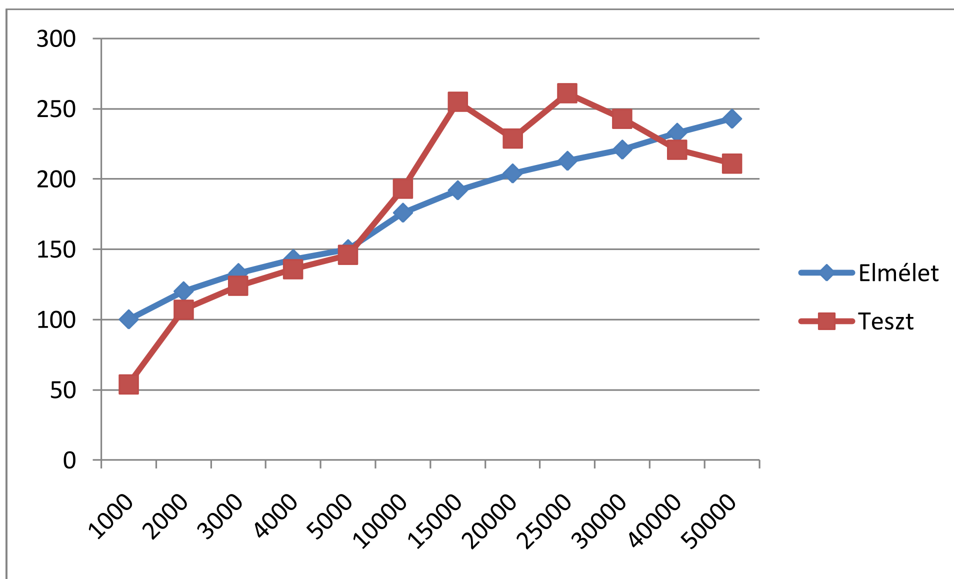
2. ábra

A 3. ábra a 20 000 csúcsra lefuttatott párhuzamos algoritmus memória telemetriáját mutatja. A memória felhasználásban mutatkozó hegyek völgyek az egyes iterációkhoz kapcsolódnak, amelyek a SCOREFIND eljárás futása során veszik fel a maximumukat. Ehhez nagyon hasonló ábrát fogunk látni később a párhuzamosított algoritmus processzor felhasználásában. Szintén érdemes megfigyelni, ahogy ezek a kiemelkedések időben egyre rövidülnek, ahogy egyre kevesebb és egyre kisebb méretű ilyen részhalmazt kell előállítani. Az utolsó, időben hosszabb kiemelkedés a tényleges függetlenséget vizsgáló eljárás nyoma.



3. ábra

Egy másik tényező, hogy hogyan teljesül az algoritmus fontos tulajdonsága, az iterációk száma. A 4. ábrán látható az iterációk száma különböző csúcsszámok mellett. A gyakorlati érték jól közelíti az elméletit.



4. ábra

Végül vizsgáljuk meg, hogy az algoritmus mely része milyen arányban foglalja a processzor erőforrásokat. Az 5. ábrán 5000 a 6. ábrán pedig 30 000 csúcsú gráfon futattam az algoritmust. Látható, hogy míg kisebb csúcsszám esetén a meghatározó eljárás a HEAVYFIND eljárás, addig nagyobb csúcsszámoknál a SCOREFIND veszi át a helyét, amely sokkal hatékonyabban párhuzamosítható! Ugyanakkor, ha eljárásbontásban nézzük meg a profil adatokat, akkor azt látjuk, hogy az összes futási idő kb. 40-50% -áért a részgráf meghatározás a felelős (bár ennek aránya is csökken a csúcsszám növelésével), aminek párhuzamosítására nincs lehetőség ebben az implementációban.

Call Tree - Method	Time [%]	Time	Invocations
main		11408 ms (100%)	1
maximalindependentset.TestConcurrentSerial.**main** (String[])		11408 ms (100%)	1
maximalindependentset.TestConcurrentSerial.**heavyFind** (org.jgrapht.Graph, java.util.Set)		6188 ms (54,2%)	155
maximalindependentset.TestConcurrentSerial.**scoreFind** (org.jgrapht.Graph, java.util.Set, java.util.Se		4177 ms (36,6%)	155
maximalindependentset.GraphUtils.**subGraph** (org.jgrapht.Graph, java.util.Set)		502 ms (4,4%)	155
maximalindependentset.TestConcurrentSerial.**indFind** (org.jgrapht.Graph, java.util.Set)		393 ms (3,4%)	155

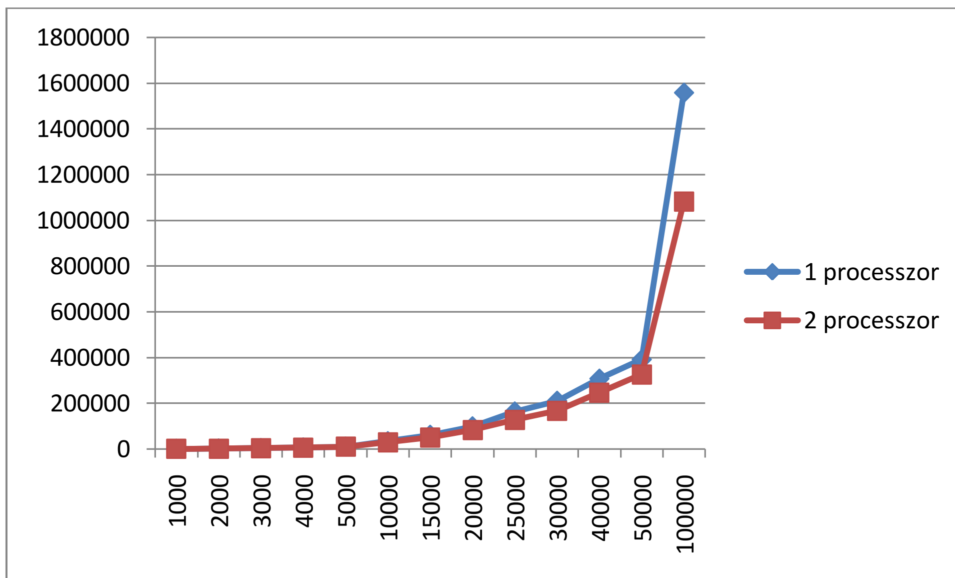
5. ábra

Call Tree - Method	Time [%]	Time	Invocations
main		219323 ms (100%)	1
maximalindependentset.TestConcurrentSerial.**main** (String[])		219323 ms (100%)	1
maximalindependentset.TestConcurrentSerial.**scoreFind** (org.jgrapht.Graph, java.util.Set, java.util.Se		122377 ms (55,8%)	326
maximalindependentset.TestConcurrentSerial.**heavyFind** (org.jgrapht.Graph, java.util.Set)		84221 ms (38,4%)	326
maximalindependentset.GraphUtils.**subGraph** (org.jgrapht.Graph, java.util.Set)		6625 ms (3%)	326

6. ábra

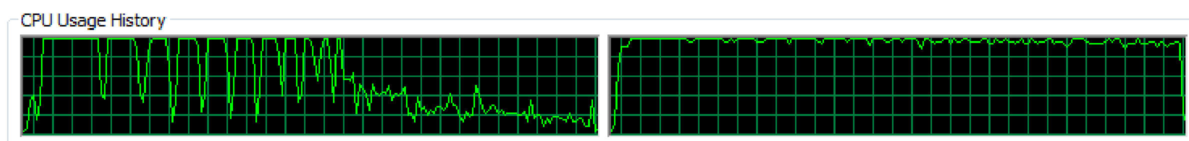
4. A párhuzamosított algoritmus

Végül vizsgáljuk meg, hogyan lehet az algoritmust párhuzamosítani. A főciklus két halmazművelete elvégezhető párhuzamosan O(log n) időben. Az INDFIND eljárás, minden élhez egy processzort rendelve szintén elvégezhető O(log n) időben. A HEAVYFIND eljárásban egy bizonyos fokszám feletti csúcsok számát határozzuk meg, ez is könnyedén párhuzamosítható csúcsszám számú processzor felhasználásával, bár a tesztek során az mutatkozott meg, hogy 2 processzoron, mivel a párhuzamosítható kódrészlet nagyon kevés munkát ad egy-egy processzornak, a párhuzamosítás rontja a futási sebességet. Ahol igazán komoly sebesség növekedés érhető el, az a SCOREFIND eljárás ciklusa, ahol minden iterációban generál egy új részhalmazt és kiszámolja a pontszámát, majd a legnagyobb pontszámút választja. Ezek az iterációk, különösen nagy csúcshalmaznál, sokáig futnak, és sokan vannak. A következő ábrán az látható, hogy a futási idő hogyan alakul az algoritmust 1, illetve 2 processzoron futtatva. Jól látható, hogy az előny nagy csúcsszámnál jön ki jelentősen (és csak a ciklus elején, amíg még a vizsgált H halmaz nagy), kis csúcsszámnál gyakorlatilag nem is jelentkezik. Ez a hatás jól látszik a 8. ábra processzor telemetriáján, ahol jól azonosíthatok az 1. processzor terheltségén az egyes ciklusok és látható az a pont, ahol már túl gyorsan fut le a SCOREFIND eljárás ahhoz, hogy a párhuzamosítás hatékonyan csökkentse a futási sebességet.



7. ábra

És végül lássuk a processzor telemetriát két processzoron való futtatás esetén:



8. ábra

A bal oldali doboz a másodlagos, a jobb oldali az elsődleges processzor kihasználtságát mutatja az idő tengelyen. Az idő tengely bal szélén kezdődik az algoritmus futása. Megfigyelhetők a bal oldali processzor kihasználtságán a SCOREFIND eljárás iterációinak elemszámmal csökkenő futási ideje. A 100% kihasználtság közötti részek, a rosszul párhuzamosítható (párhuzamosított) többi lépés által hagyott űrt jelentik. A futási időben középtájon megszűnik a másodlagos processzor kihasználtsága. Ez körülbelül 10 000 elemnél következik be, ez az a pont, ami alatt már nem érdemes párhuzamosan futtatni az egyes iterációkat. Ez a pont a 7. ábrán is megfigyelhető.