Gyuris Bence

N-Test szimuláció

nagyteljesítményű számítógépen

Nagyteljesítményű párhuzamos feldolgozás

Házi feladat dokumentáció

2015 Ősz

Tartalom

1. Feladatspecifikáció

A feladatom egy N-Test szimulációt futtató program megvalósítása volt. Ez a szimuláció a fizikában és a csillagászatban használt, N darab testből álló dinamikus rendszert szimulál, amelyben a testek mozognak és közöttük bizonyos fizikai hatások lépnek fel.

A feladatom során egy a naprendszerhez hasonló **2 dimenziós** rendszert szimuláltam. Minden testnek van x és y pozíciója, (v_x, v_y) sebességvektora és m tömege (ha m tömeg -1, akkor a test nem megfigyelt már). A testek egymásra gravitációs erővel hatnak, valamint ha keresztezik egymás útját, akkor az impulzusmegmaradásnak megfelelően ütköznek (egyesülnek). A távolság mértékegysége 10^8m, a tömegé 10^{24}kg, az időé 1 nap, a gravitációs konstans ehhez van igazítva. Így könnyen lehet számolni naprendszer nagyságrendben.

A megoldásomban kezdetben a rendszer középpontjában 1 csillag áll ($2x10^{30}$kg), van 100 óriásbolygó ($2x10^{27}$kg) és 2899 bolygó ($6x10^{24}$kg), tehát összesen 3000 test van. A bolygók a csillag körül helyezkednek el véletlenszerűen, normáleloszlással. A bolygók a csillag körül körpályán mozognak, tehát a kezdeti sebességük az ebből adódó kerületi sebesség. A bolygók tömegében és sebességében van még egy random faktor is normáleloszlással, hogy az eredmény ne legyen kiszámítható.

A program ezután szimulál a bemeneti paraméterben kapott számú napot (alapértelmezetten 365 nap). Minden ciklusban frissíti a testek sebességét/pozícióját a vizsgált fizikai törvényeknek megfelelően, majd a rendszer állapotát az eredményfájlba menti.

2. Fejlesztői dokumentáció

Az elkészült szoftvert a következő fejezetben jellemzem. A szoftver forrása és binárisa megtalálható ezen dokumentummal egy állományban. Az elkészült szoftver két részből áll, a szimulációt végző, szerveren futó C++-ban megírt programból és a felhasználónál futó eredmény fájlban tárolt naprendszer idővonalát lejátszó C# alkalmazásból.

2.1 Szimulátor

Az első rész a C++-ban megírt szimulációt elvégző program. Ennek a forrása a *main.cpp* fájlban és az *intersection.h* fájlban találhatóak a *source* mappában. A kódban megvalósított függvényeket a következő táblázatokban foglalom össze.

intersection.h

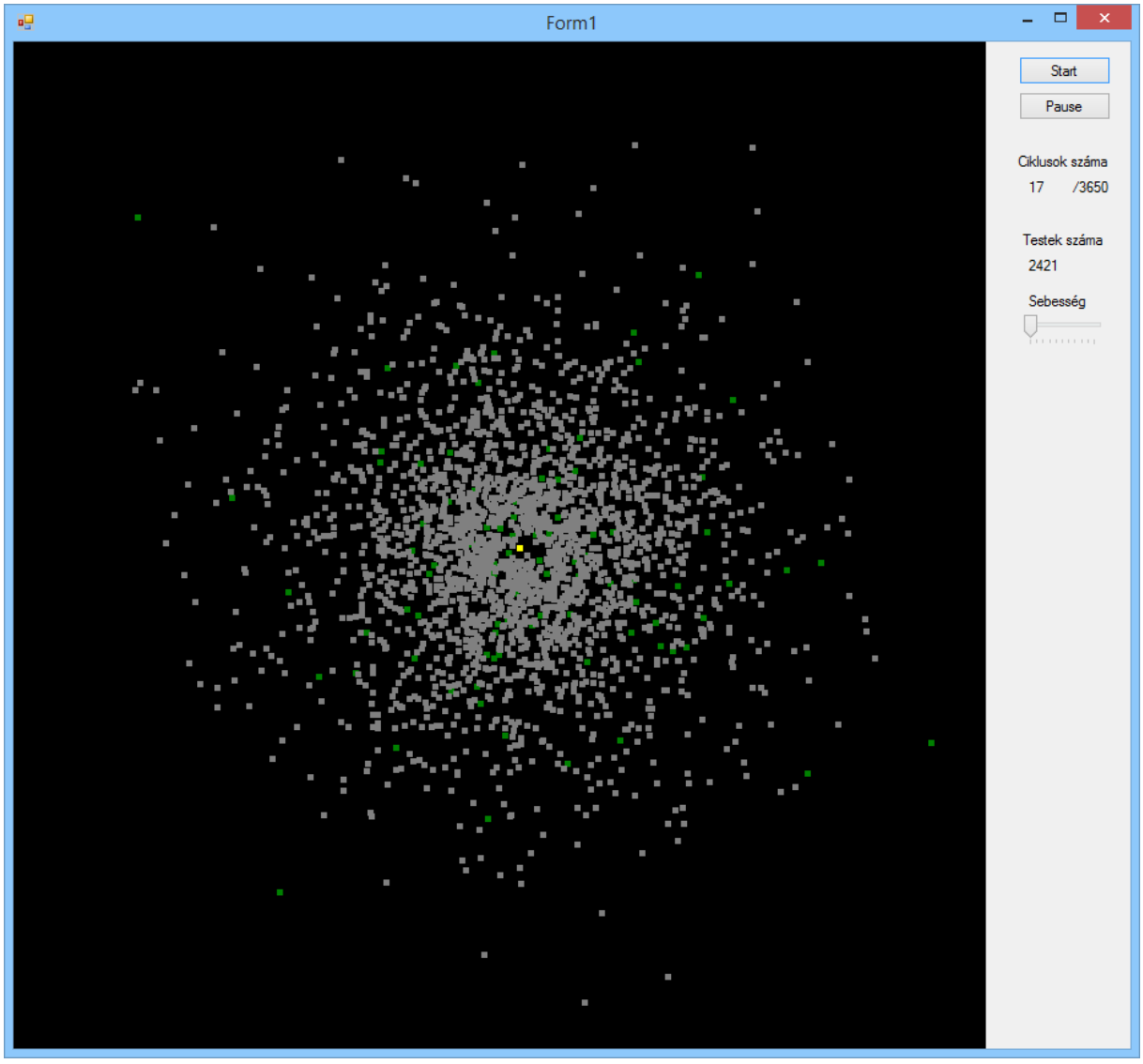
doIntersect	Igazzal tér vissza, ha a paraméterül kapott 2 szakasz (kezdő és végpontok alapján megadva) metszi egymást. Ez az egyetlen kívülről használt függvény.

main.cpp

main	A program belépési pontja. Feldolgozza a paraméterül kapott értéket (napok száma), létrehozza a kimeneti fájlt, méri a futási időt, inicializálja az állapotteret majd lefuttatja a szimuláció összes napját. Minden nap állapotát menti az eredmény fájlban. A szimuláció végén kiírja a futás idejét és bezárja az eredmény fájlt.
SimulateDay	Szimulál egy napot. Először kiszámolja a sebességüknek megfelelően minden test új pozícióját, majd a rá ható N db. gravitációs erő összege alapján kiszámolja az új sebességét (gyorsulás), végül ellenőrzi, hogy nem hagyta-e el az ellenőrzött tartományt (a csillagrendszert). Ezután ellenőrzi, hogy történtek-e az adott napon ütközések a CheckCollusion függvénnyel.
CheckCollusion	Minden lehetséges test párt megvizsgál, ha a testek túl közel vannak egymáshoz vagy keresztezik egymás útját, akkor az egyik tömegét -1-be állítja, a másiknak pedig az impulzusmegmaradásnak megfelelő sebességet adja és növeli a tömegét.
InitializeSpace	Beállítja a testek kezdeti pozícióját, sebességét és tömegét a program paramétereinek megfelelően.
minDistance	Megadja azt a távolságoz, amelynél ha két test közelebb van egymáshoz, akkor ütközniük kell. A távolság kiszámításához szükséges a testek tömege.
outOfSpace	Igazzal tér vissza ha a paraméterül kapott test vagy koordináta a vizsgált tartományon (naprendszeren) kívülre esik.

2.2 Megjelenítő, felhasználói felület

A program második része „lejátsza” a szimulátor által generált *timeline.dat* eredményfájlt, így a felhasználó vizuálisan elemezni tudja a szimulált eseményeket. Ennek a programnak a forrása a GalaxySimulatorViewer mappában található meg, ez egy VS2013-mas sollution. A lefordított binárist a GalaxySimulatorViewer.exe tartalmazza.



Ez a program egy egyszerű C#-ban megírt Windows Forms-os alkalmazás. Egy ablakot tartalmaz ezt a Form1 fájlokban definiáltam. A jobb oldali Start gombban indítható a lejátszás, a sebesség csúszkával módosítható a lejátszás gyorsasága.

button1_Click: Eseménykezelő, itt történik az állapottér kirajzolása. Megnyitja a *timeline.dat* fájlt, végigmegy az összes napon. Először beolvassa a testek állapotát, majd kirajzolja a testeket a felületen négyzetként. Sárga színnel rajzolja a napot, zölddel az óriásbolygókat és szürkével a normál bolygókat. Egy ciklus kirajzolásának a végén a csúszka állapotától függő ideig várakozik, mielőtt kirajzolja a következő ciklus állapotát. Így láthatjuk a bolygókat a szimulációnak megfelelő mozgás közben.

2.3 Fordítás, telepítés, futtatás

A lejátszó programot Visual Studio 2013 segítségével lehet fordítani Windows rendszereken. A *GalaxySimulatorViewer/GalaxySimulatorViewer.sln* fájlt kell megnyitni.

A szimulátor főprogramot a para gépeken a következő paranccsal lehet fordítani:

```
g++ -fopenmp main.cpp -o nbody -lgomp -std=c++11
```

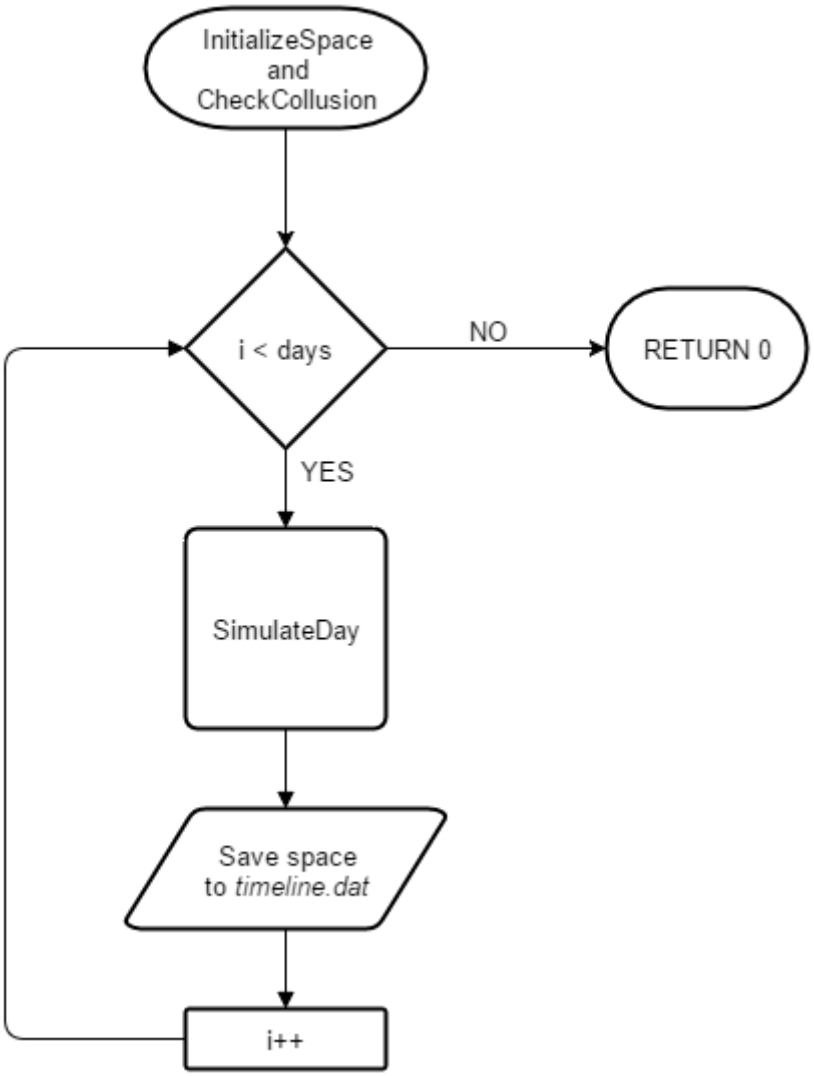
Ezután a következő paranccsal lehet szimulálni pl. 10 éves időtartamot:

```
./nbody 3650
```

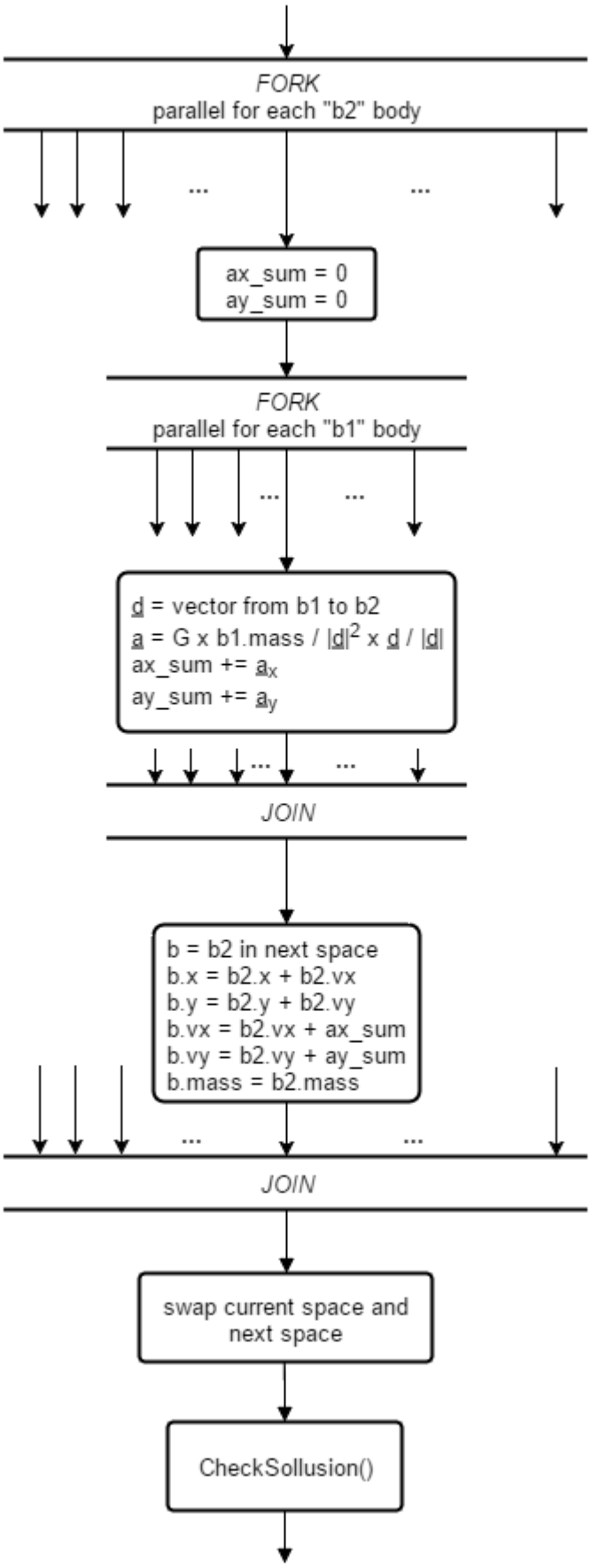
Ekkor a program generálni fogja a *timeline.dat* eredményfájlt. Ezt le kell tölteni a szerverről egy Windows-os gépre és be kell másolni a GalaxySimulatorViewer.exe programmal egy mappába. Ekkor a megjelenítő programot elindítva lejátszhatjuk a szimulációt, megnézhetjük a testek mozgását.

3. Az algoritmus

A szimulátor futásának algoritmusa a napok szimulálásából azon belül a gravitációs erők hatásának és az ütközések szimulálásából áll. A program legkülső ciklusának algoritmusa a következő ábrán látható:



Tehát sorosan fut bizonyos számú SimulateDay művelet és ezek között az állapotot mindig elmentjük. A SimulateDay a rendszer egy napjának lefutását szimulálja, a működése a következő ábrán látható:



Minden b2 testre kiszámolom a gravitációs gyorsulását. Ezek a számítások egymástól függetlenek, ezért párhuzamosan is történhetnek.

Minden b2 testhez vizsgálom a rendszer minden egyéb b1 testét (párhuzamosan).

Kiszámolom b1 és b2 távolságát. Ez alapján és a b2 tömeg és a gravitációs konstans alapján már számolható a b1 által okozott gravitációs gyorsulás b2-ben. Ezeket a gyorsulásokat (párhuzamosan) összegzem hogy megkapjam az eredendő gyorsulást (ax_sum, ay_sum).

Kiszámolom a test következő állapotának adatait. Kiszámolom az új pozícióját a sebességétől függően. Kiszámolom az új sebességét az eredendő gyorsulásának függvényében.

Beállítom a testeket a kiszámolt új adataival.

Ellenőrzöm, hogy történtek-e a napon ütközések. Ennek a függvénynek a kifejtését a következő pszeudo-algoritmus tartalmazza.

megjegyzés: A Fork és Join-ok közötti szakaszok párhuzamosítását az OpenMP keretrendszer végzi el.

A CheckCollusion függvény megértését a legegyszerűbben pszeudo-kódjával tudom segíteni. A belső ciklus párhuzamosítását itt is az OpenMP keretrendszer végezte el. A belső ciklus itt is egy párhuzamos összegzés, az egyes iterációk az összegzésen kívül egymástól függetlenek.

```
for i = 0..(N-1) do
      b1 = space[i]
      mass_sum = b1.mass
      vxMmass_sum = b1.mass*b1.vx
      vyMmass_sum = b1.mass*b1.vy
      parallel for k = (i+1)..N do
            b2 = space[k]
            if (b1 is close to b2 or paths are crossing) do
                  mass_sum += b2.mass
                  vxMmass_sum += b2.mass*b2.vx
                  vyMmass_sum += b2.mass*b2.vy
                  remove b2
            od
      od
      b1.mass = mass_sum
      b1.vx = vxMmass_sum / mass_sum
      b1.vy = vyMmass_sum / mass_sum
od
```

Az algoritmus során minden testhez megvizsgálom az őt követő testeket, azoknak amelyekkel ütközik összegzem a tömegét és a sebesség-tömeg szorzatát és törlöm őket, majd növelem az első test tömegét és változtatom a sebességét az impulzusmegmaradás törvénye szerint.

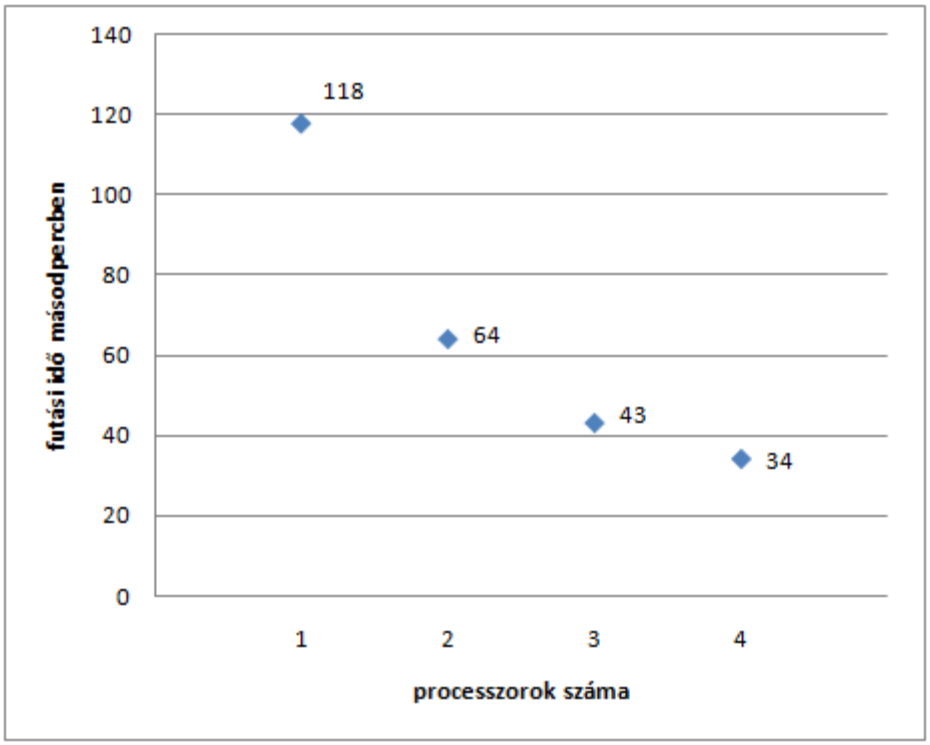
4. Eredmények

A programot a para.iit.bme.hu szerver node-jain futtattam. Ezen a szerveren maximum 4 processzor érhető el. Elméletileg az algoritmusomban N test esetén N darab processzor lenne hatékonyan kihasználható futás közben (gravitációs erő méréséhez akár N^2 is). A speed-up pontos méréséhez futtattam 4, 3, 2 és 1 párhuzamosan működő processzoron a következő parancsokkal:

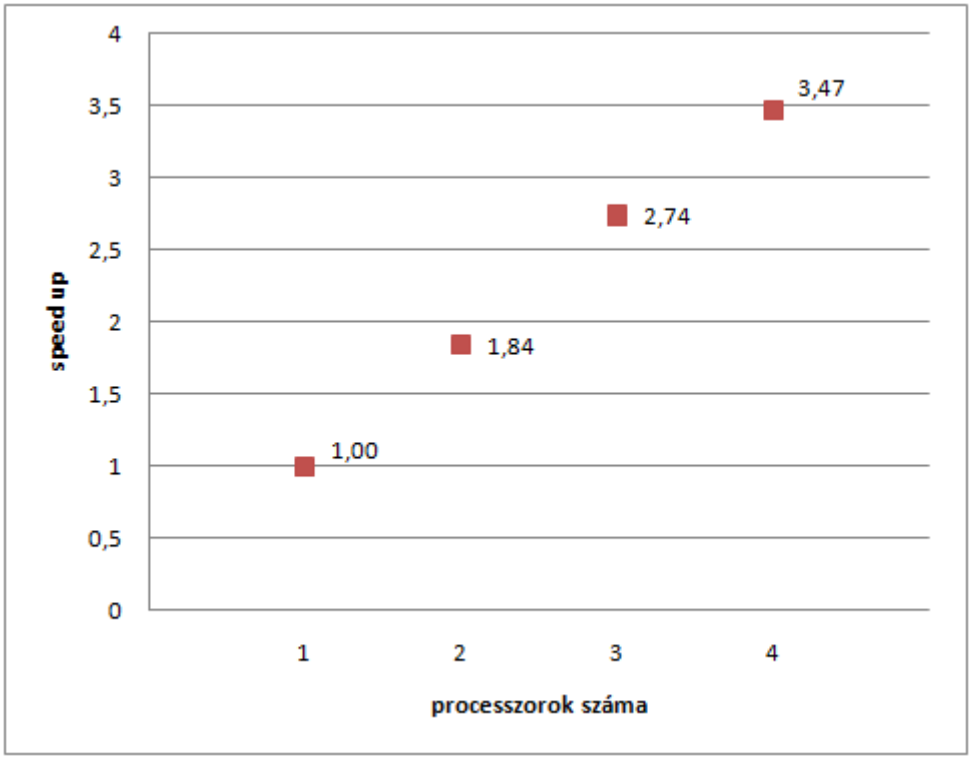
- `sbatch -n 4 -o out4.txt run.sh 4 ./nbody`
- `sbatch -n 3 -o out3.txt run.sh 3 ./nbody`
- `sbatch -n 2 -o out2.txt run.sh 2 ./nbody`
- `sbatch -n 1 -o out1.txt run.sh 1 ./nbody`

A *run.sh* fájl megtalálható a dokumentum mellett csatolva.

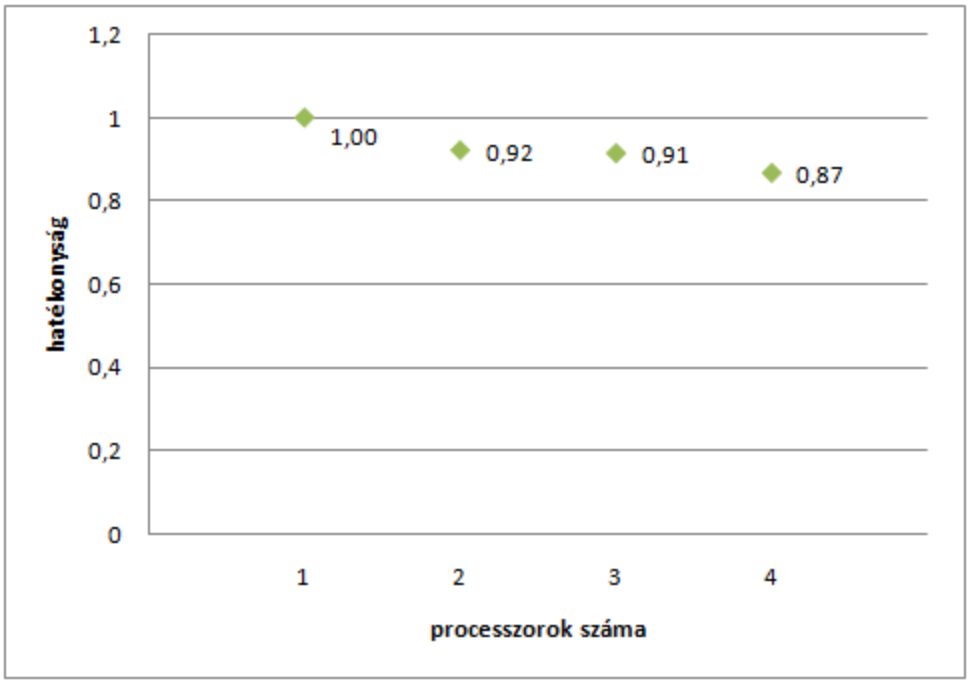
A programok kapott futási idejeit a következő ábrán szemléltetem:



Az ezekből számított speed up-okat (sebességnövekedést) a következő ábrán jelenítettem meg:



Végül a futások hatékonysága (efficiency: En = Sn/N):



Tehát a processzorszám növekedésével a hatékonyság törvényszerűen romlik, de érzékelhető a sebességnövekedés, még 4 processzornál is.