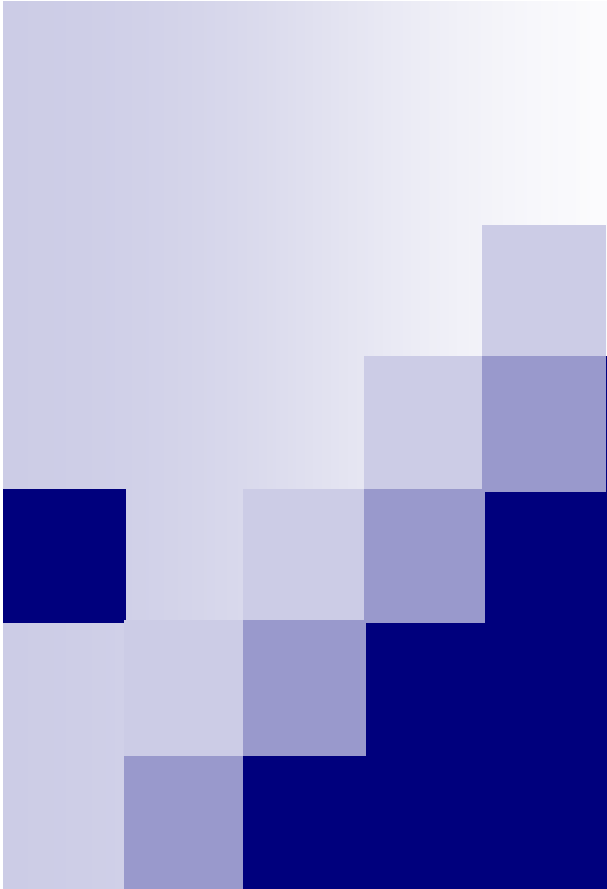


Párhuzamos programozási feladatok

BMF NIK

2008. tavasz

B. Wilkinson és M. Allen oktatási
anyaga alapján készült



Gravitációs N-test probléma

Fizikai törvények alapján testek
helyzetének, mozgásjellemzőinek és a
rájuk ható erőknek a meghatározása.
Rekurzív szétosztás



Gravitációs N-test probléma - egyenletek

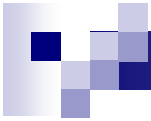
Az m_a és m_b tömegű testek közötti gravitációs erő:

$$F = \frac{m_a * m_b * g}{r^2}$$

g a gravitációs gyorsulás, r a két test távolsága. Az F erő Newton 2. törvénye alapján gyorsítja a testet:

$$F = m * a$$

m a test tömege és a a gyorsulás.



Az időintervallumot jelölje Δt . Az m tömegű testre ható erő:

$$F = m * \frac{(v^{t+1} - v^t)}{\Delta t}$$

Az új sebesség így:

$$v^{t+1} = v^t + \frac{F * \Delta t}{m}$$

ahol v^{t+1} a sebesség a $t + 1$ időpillanatban és v^t a sebesség t időpontban.

A Δt idő alatt a helyzetváltozás:

$$x^{t+1} - x^t = v * \Delta t$$

ahol x^t a pozíció t paraméter szerint.

A test mozgása során az erő változik. A számítást iterálni kell.



N-test probléma soros kódja

```
for (t = 0; t < tmax; t++)           /* minden időperiódusra */
    for (i = 0; i < N; i++) {        /* minden testre */
        F = Force_routine(i);        /* az i. testre ható erő */
        v[i]new = v[i] + F * dt / m; /* az új sebesség */
        x[i]new = x[i] + v[i]new * dt; /* és pozíció */
    }

for (i = 0; i < N; i++) {            /* minden testre */
    x[i] = x[i]new;                  /* a sebesség és a helyzet */
    v[i] = v[i]new;
}
```

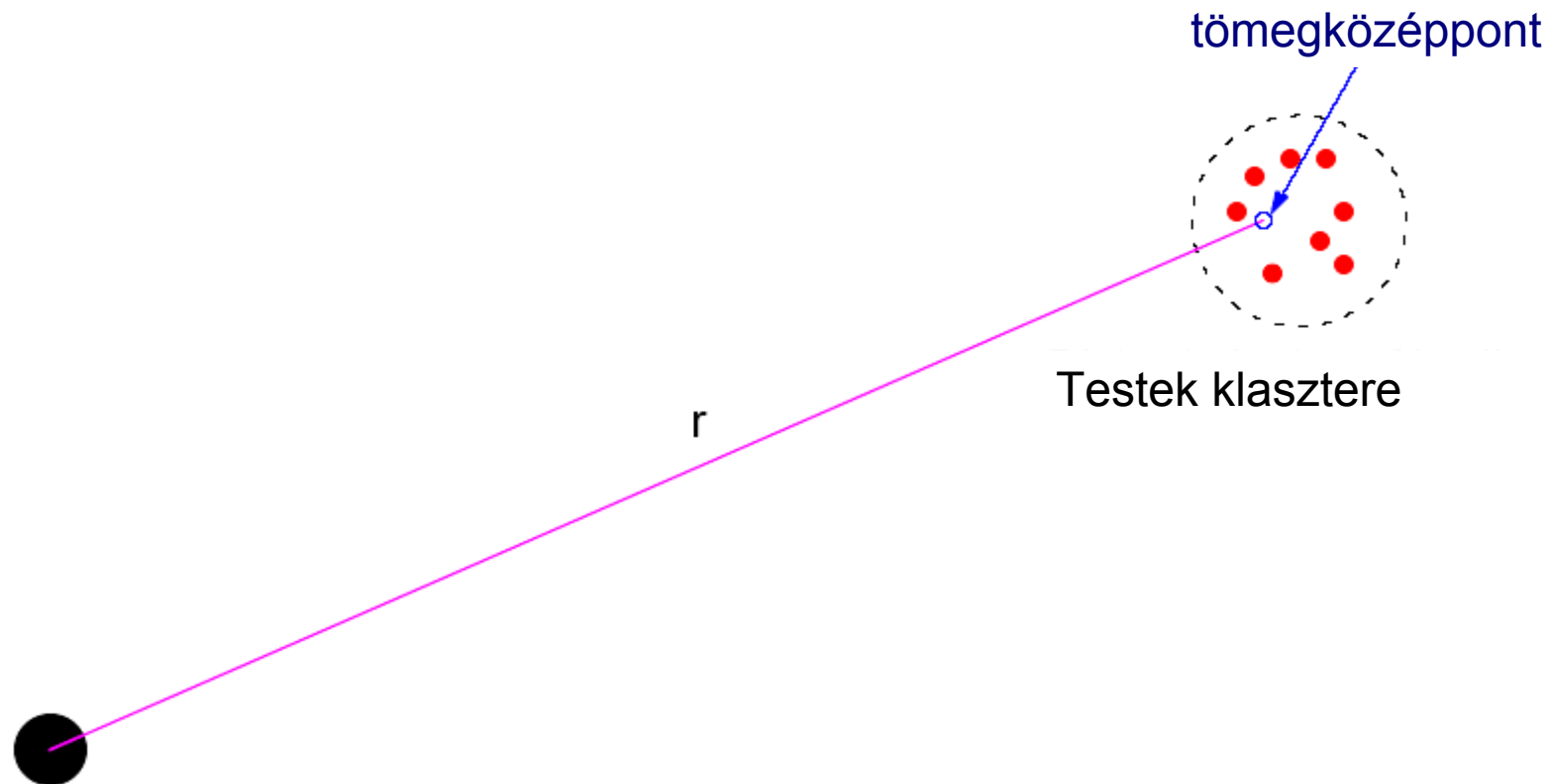


Párhuzamos kód indoka

A soros program $O(N^2)$ nagyságrendű egy iterációban, mivel mind az N test $N - 1$ testre fejt ki erőt.

Ha N nagy, akkor az N -test problémára nem hatékony a soros kód.

A komplexitás csökkenhető, ha a távoli testek csoportját egy tömegbe redukáljuk úgy, hogy a klasztert alkotó testek tömegközéppontjába transzformáljuk az össztömegeket:





Barnes-Hut algoritmus

Induljunk ki a teljes térből, ahol egy kocka tartalmazza a testeket.

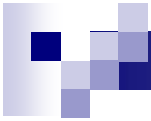
- Elsőként osszuk a kockát nyolc részkockára.
- Ha egy részkocka nem tartalmaz testet, akkor a továbbiakban ne vegyük figyelembe.
- Ha a részkocka egy testet tartalmaz, akkor megőrizzük.
- Ha egy részkocka egynél több testet tartalmaz, akkor rekurzívan addig részkockákra bontjuk, amíg nem egy test lesz benne.



Készítsünk egy nyolcas fát (*octtree*) – ahol minden csomópontból maximum nyolc él indul ki.

A levelek cellákat reprezentálnak, amelyek pontosan egy testet tartalmaznak.

A fa konstruálása után a részkocka teljes tömegét, és a tömegközéppont koordinátáját tároljuk minden csomópontban.



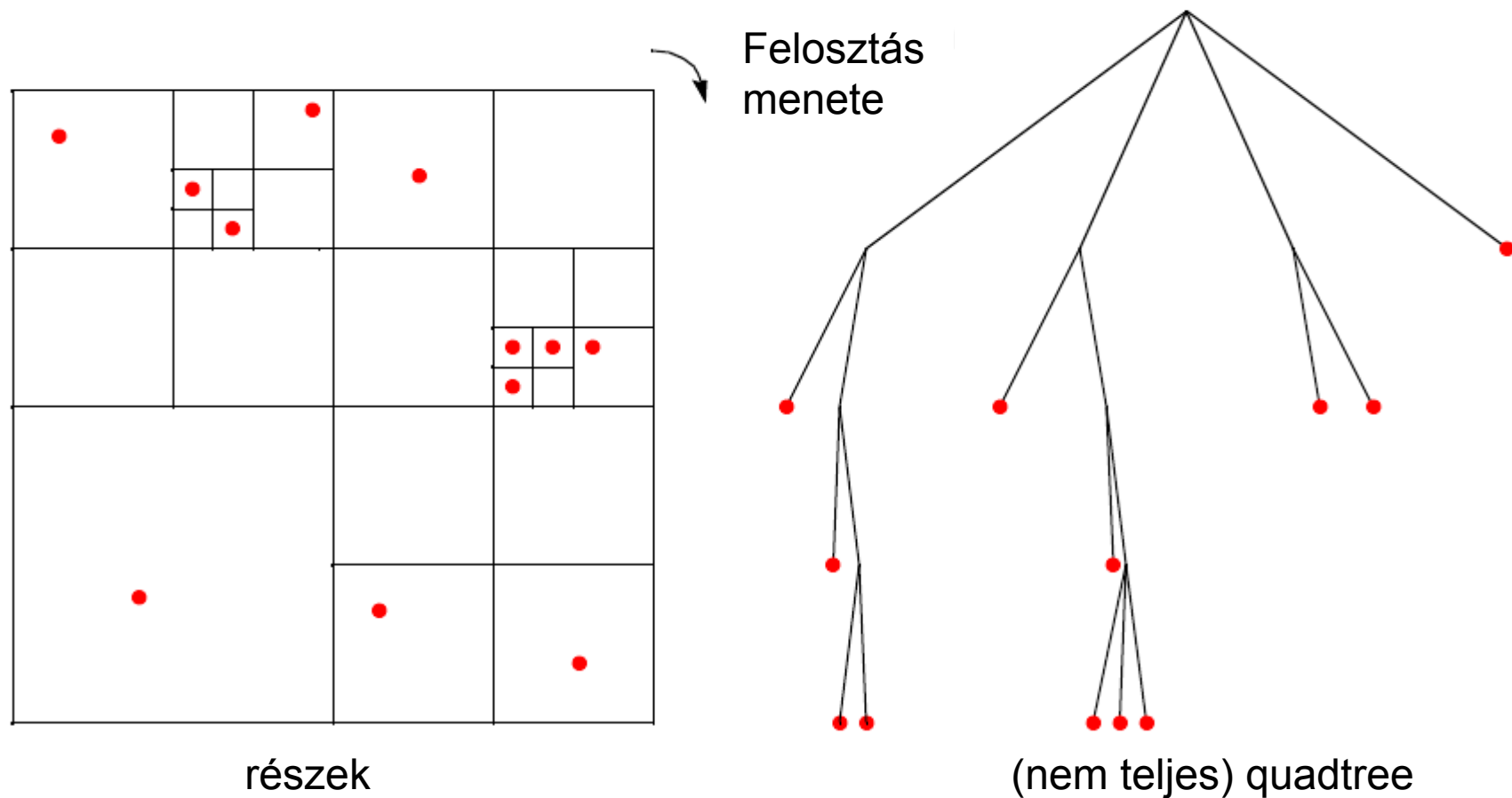
Minden testre ható erő megkapható, ha a fa gyökerétől elindulunk és eljutunk a csomóponting, ahol a klaszteres közelítés használható, azaz amikor:

$$r \geq \frac{d}{\theta}$$

ahol θ konstans tipikusan egy, vagy annál kisebb.

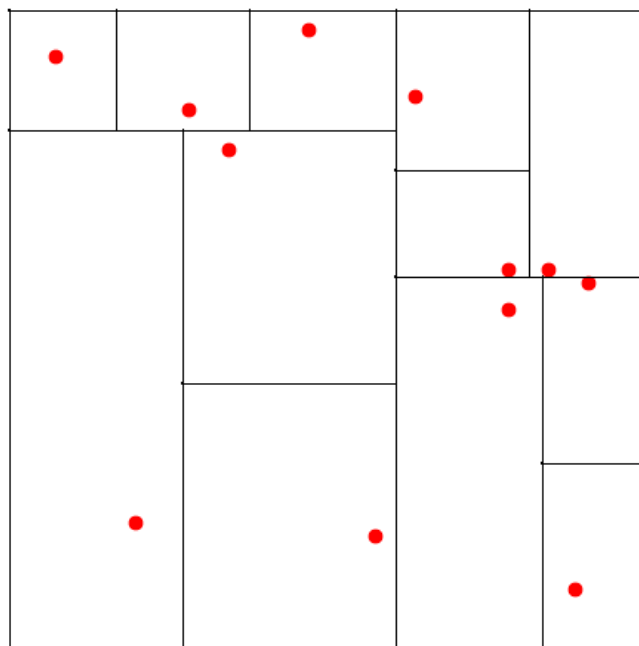
A fa elkészítése $O(n \log n)$ nagyságrendű időt vesz igénybe, és az erőké is, így a teljes idő $O(n \log n)$.

2 dimenziós tér rekurzív felosztása



Ortogonalis rekurzív kettéosztás

2D-s területben elsőként keressünk egy függőleges egyenest, amely úgy osztja két részre, hogy mindkét félbe azonos számú test essen. Minden így kapott részt vízszintesen osszunk ketté egy egyenessel, hogy azonos számú testet tartalmazzanak. Amíg szükséges folytassuk a szétosztásokat.





Alacsony szintű képfeldolgozás

Nyilvánvalóan párhuzamosítható számítás.
Számos alacsony szintű képfeldolgozási
művelet csak lokális adaton dolgozik, vagy
esetleg egy szűk környezetből veszi inputját.



Geometriai transzformációk

- Objektum eltolása Dx értékkel x irányba és Dy értékkel y irányba:

$$x' = x + Dx$$

$$y' = y + Dy$$

ahol x és y az eredeti koordináták, x' , valamint y' az újak.

- Objektum skálázása x irányban S_x és y irányban S_y értékkel:

$$x' = x * S_x$$

$$y' = y * S_y$$



Geometriai transzformációk

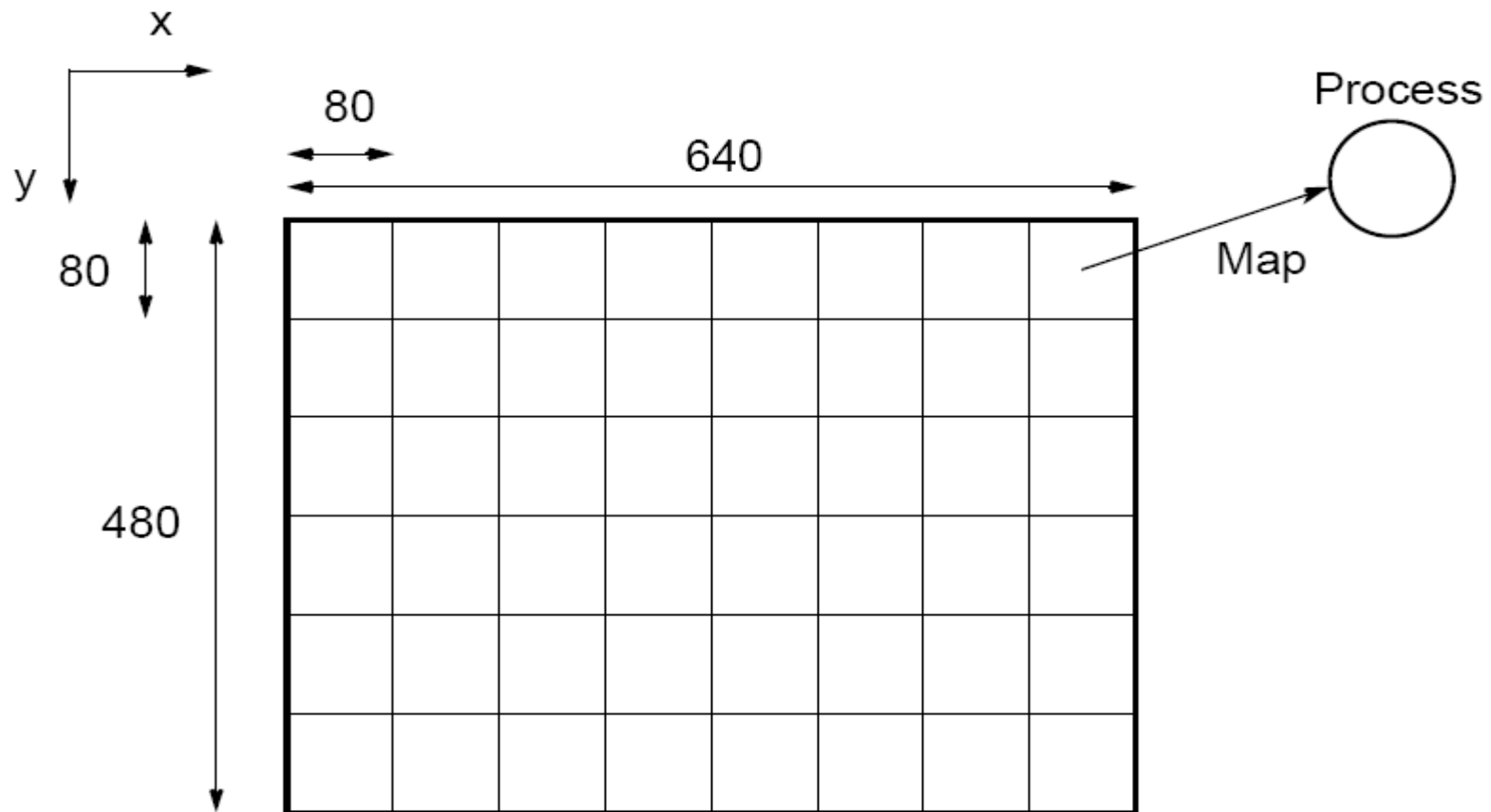
- Objektum forgatása θ szöggel a koordinátarendszer z tengelye körül:

$$x' = x * \cos \theta + y * \sin \theta$$

$$y' = -x * \sin \theta + y * \cos \theta$$

Régiókra particionálás

- Minden régiót külön processzben számolhatunk





Mandelbrot halmaz

Nyilvánvalóan párhuzamosítható
számítás



Mandelbrot halmaz

- A komplex sík pontjai, amelyek iteráció során változtatunk, de egy határon belül maradnak.

$$z_{k+1} = z_k^2 + c$$

ahol z_{k+1} a $(k+1)$ -dik iterációja a $z = a + bi$ komplex számnak és c komplex szám a pont kezdeti helyzetét adja meg a komplex síkon. z kezdeti értéke nulla.

- Az iteráció addig folytatódik, amíg z nagysága kettőnél nem nagyobb, vagy az lépések száma egy határt el nem ér. A komplex z szám nagysága, mint vektorhossz:

$$z_{length} = \sqrt{a^2 + b^2}$$

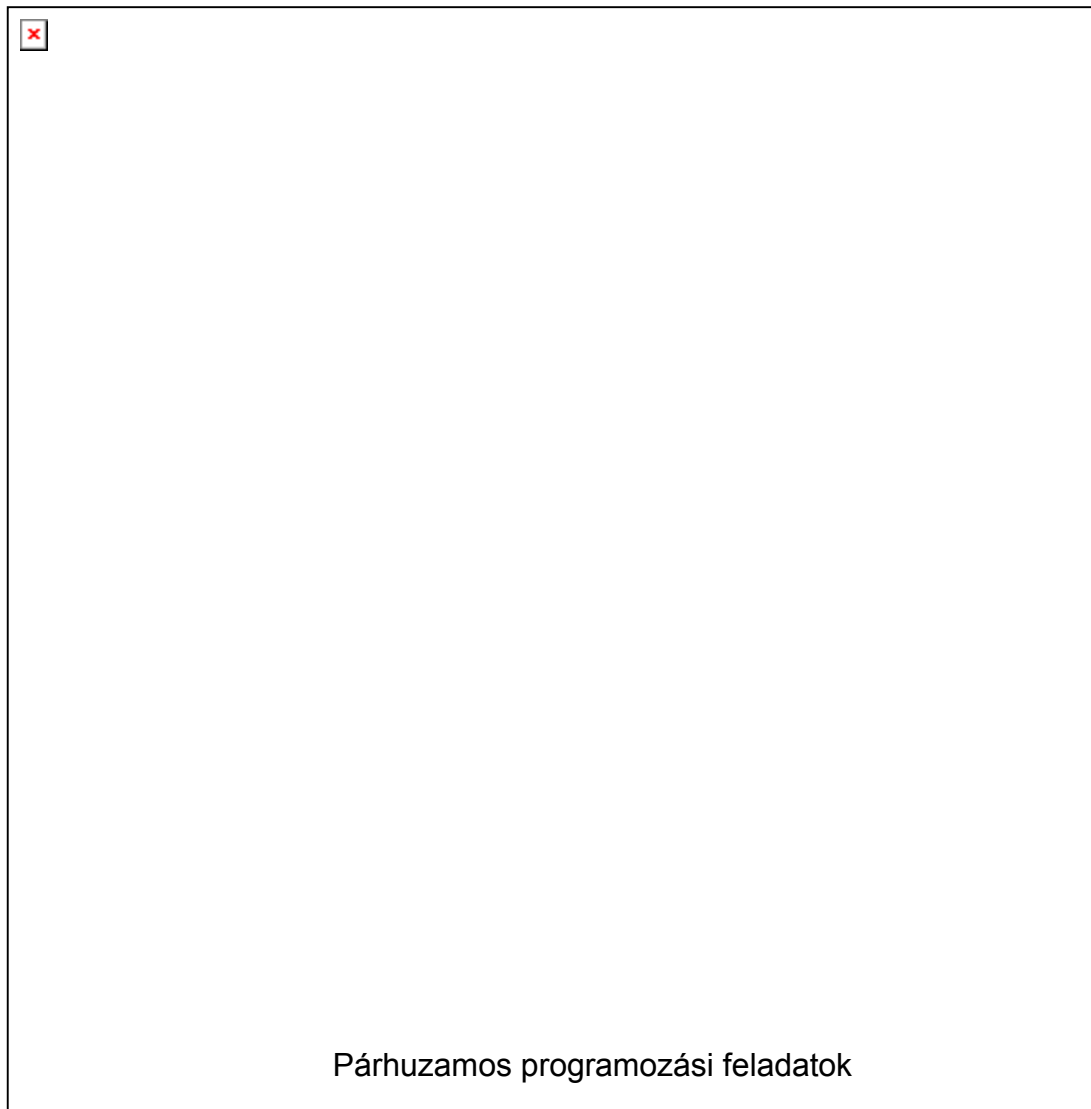


Soros rutin egy pontra

```
structure complex {
    float real;
    float imag;
};
int cal_pixel(complex c)
{
    int count, max;
    complex z;
    float temp, lengthsq;
    max = 256;
    z.real = 0; z.imag = 0;
    count = 0;                                /* number of iterations */
    do {
        temp = z.real * z.real - z.imag * z.imag + c.real;
        z.imag = 2 * z.real * z.imag + c.imag;
        z.real = temp;
        lengthsq = z.real * z.real + z.imag * z.imag;
        count++;
    } while ((lengthsq < 4.0) && (count < max));
    return count;
}
```



Mandelbrot halmaz



Párhuzamos programozási feladatok



Párhuzamosítás

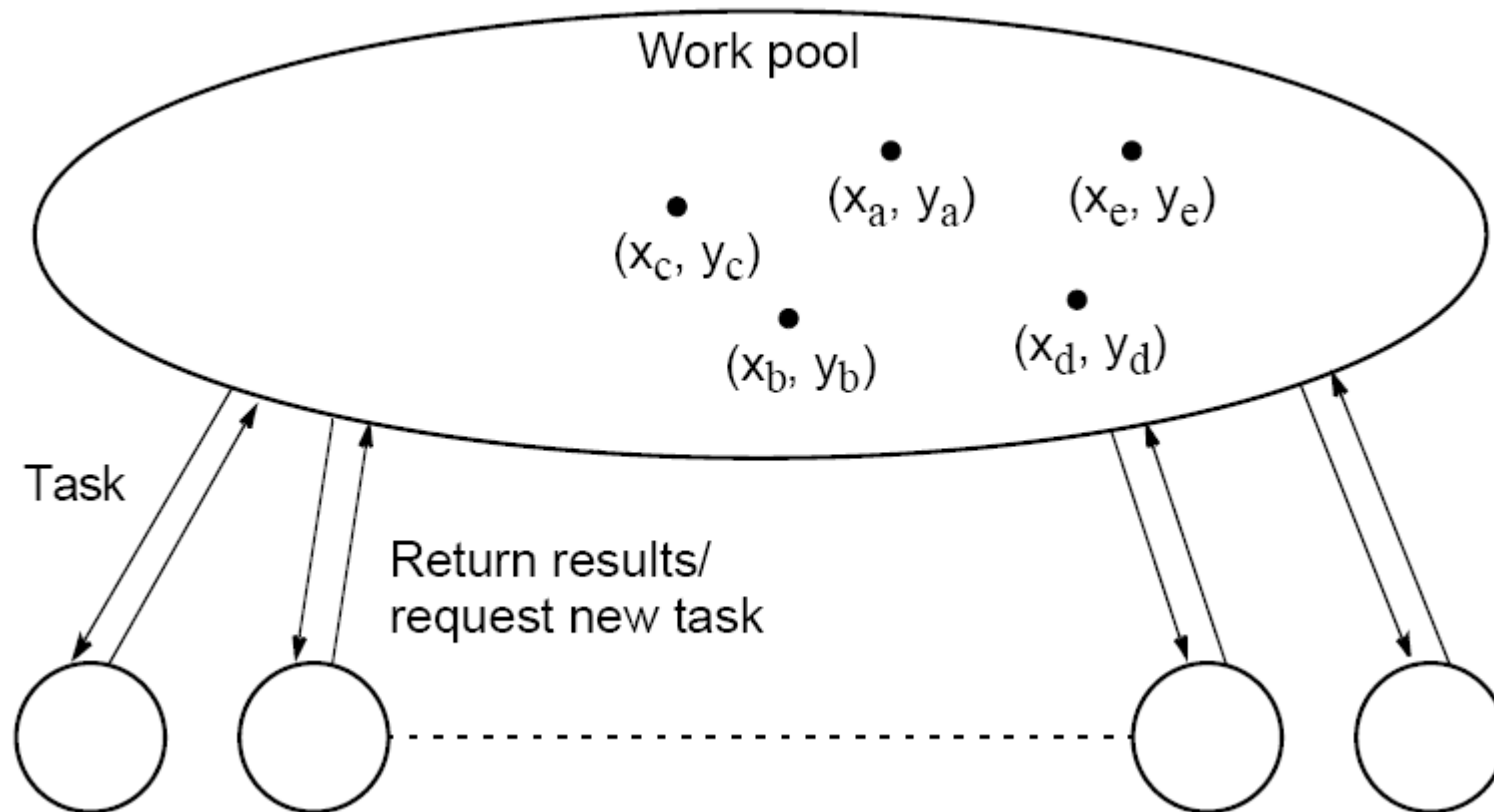
■ Statikus feladat kiosztás

- ☐ Egyszerűen osszuk a teret fix számú részre és minden részt külön processz számoljon.
- ☐ Nem igazán hatékony, mert minden régió eltérő számú iterációt igényel.

■ Dinamikus feladat kiosztás (Work pool modell)

- ☐ Amikor a processz elkészül a korábbi számítással, új feladatot kap.

Work pool modell





Celluláris automata

Számítások randevú típusú
szinkronizálással



Celluláris automata

- A feladatteret cellákra osztjuk.
- Minden cella véges állapotok közül pontosan egyet vesz fel.
- A cellákra a szomszédjai valamilyen hatással vannak bizonyos szabály szerint, és minden cella egy „generációs” szabállyal is rendelkezik, amely szimultán fut.
- A szabályok újra és újra alkalmazásra kerülnek minden generációs lépésben, így a cellák fejlődnek, vagy megváltoztatják állapotukat generációról generációra.

A legismertebb celluláris automata John Horton Conway cambridge-i matematikus „Élet játéka” (“Game of Life”).



Az élet játéka

- Táblás játék – elméletben végtelen méretű kétdimenziós cellatömbbel. Minden cellában egy organizmus lehet, nyolc szomszédja van. Kezdetben néhány cella foglalt.
- A következő szabályokat alkalmazzuk:
 1. Minden organizmus, amelynek két, vagy három szomszédja van, életben marad a következő generációra.
 2. Minden organizmus, amelynek négy, vagy több szomszédja van, meghal a túlnépesedés miatt.
 3. Minden organizmus, amelynek maximum egy szomszédja van, elpusztul az izoláltság miatt.
 4. Minden üres cellában, amelynek pontosan három nem üres szomszédja van egy új organizmus születik.

E szabályokat Conway határozta meg hosszas kísérletek után.



Más egyszerű példa* – halak és cápák

- Az óceán három dimenziós tömbbel modellezhető
- Minden cella tartalmazhat vagy halat, vagy cápát
- A halak és a cápák szabályokat követnek

*: Más példák is lehetnek: nyulak – rókák



Halak

- A következő szabályok szerint élnek:
 1. Ha valamely szomszédos cella üres, akkor odaúszik.
 2. Ha több szomszédos cella üres, akkor véletlenszerűen választ egyet.
 3. Ha nincs üres szomszédos cella, akkor helyben marad.
 4. Ha a hal mozog és eléri nemzési idejét (paraméter), akkor egy bébihalnak életet ad, amely a megüresedő cellába kerül.
 5. A hal x generáció után meghal.



Cápák

- A következő szabályok vonatkoznak rájuk:
 1. Ha valamely szomszédos cellában hal van, akkor a cápa abba a cellába megy és megeszi a halat.
 2. Ha több szomszédos cellában van hal, akkor a cápa véletlenszerűen választ egyet, abba a cellába megy és megeszi a halat.
 3. Ha nincs hal a szomszédos cellában, akkor a cápa egy nem foglalt szomszédos cellába mozog olyan szabályok szerint, mint a hal.
 4. Ha a cápa mozog és eléri nemzési idejét, akkor egy bécicápának életet ad, amely a megüresedő cellába kerül.
 5. Ha a cápa y generáción keresztül nem eszik, akkor meghal.



Valós CA példák

- Folyadék/gáz dinamika
- Folyadékok és gázok áramlása testek körül
- Gázok diffúziója
- Biológiai fejlődés
- Légáramlás repülőgép légcsavarjánál
- Homok eróziója/mozgása folyóparton



Részben szinkron számítás

- Nem minden iterációnál végezzük el a szükséges szinkronizációt
- Ez jelentősen csökkentheti az átfutási időt, mert a randevú típusú szinkronizálás általában nagyon lelassítja a számításokat
- A globális szinkronizációt randevú rutinnal végezzük bizonyos időnként