Modelarea unui automat celular (Game of Life)

Responsabili: Catalin Leordeanu (catalin.leordeanu@cs.pub.ro), Alecsandru Patrascu (alecsandru.patrascu@gmail.com)
Deadline: 06.11.2016, ora 23:55

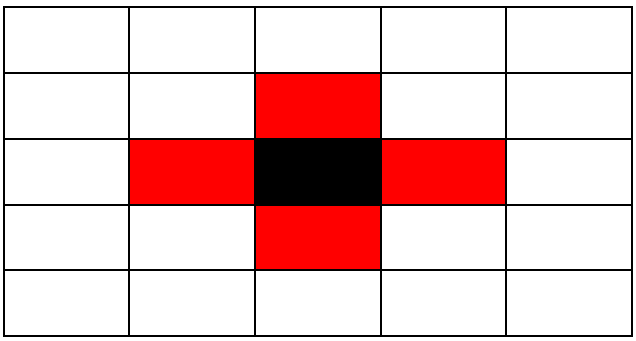
1. Introducere

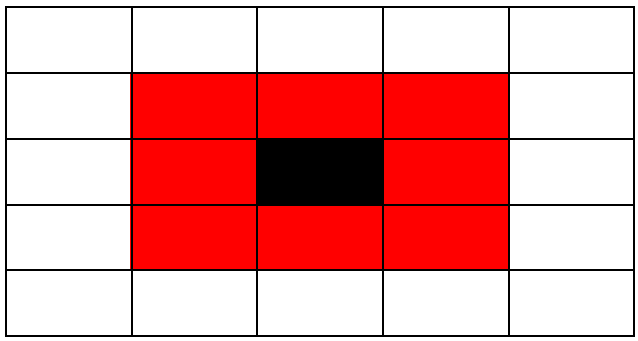
Un automat celular este un instrument important folosit in stiinta, ce are drept scop modelarea unor varietati de fenomene naturale. Acestea se numesc asa deoarece produc o serie de efecte unor unitati individuale, numite celule.

Unul dintre cele mai importante astfel de automate este cel creat de John Conway, numit “Game of Life”. Conway a dorit sa modeleze automatul astfel incat sa se poata observa un comportament evolutiv. Astfel, modelul initial avea sa creasca si sa se evolueze intr-un alt model, de cele mai multe ori avand rezultate care nu puteau fi anticipate. De asemenea, acesta a dorit ca fiecare model din automat sa nu ramana fix, ci sa aiba posibilitatea sa se disipe sau sa-si modifice forma.

Prin urmare, automatul este vazut in plan ca o matrice. Are o stare initiala de plecare si un numar determinat de pasi. La fiecare pas, orice celula are una din doua stari: in viata (ALIVE) sau moarta (DEAD).

Ca orice alt automat celular, starea unei celule este determinata de starea vecinilor acesteia. Mai jos se pot vedea doua astfel de exemple, in care pentru celula de culoare neagra se considera 4 sau 8 vecini. In cazul nostru ne intereseaza toti cei 8 vecini ai unei celule.

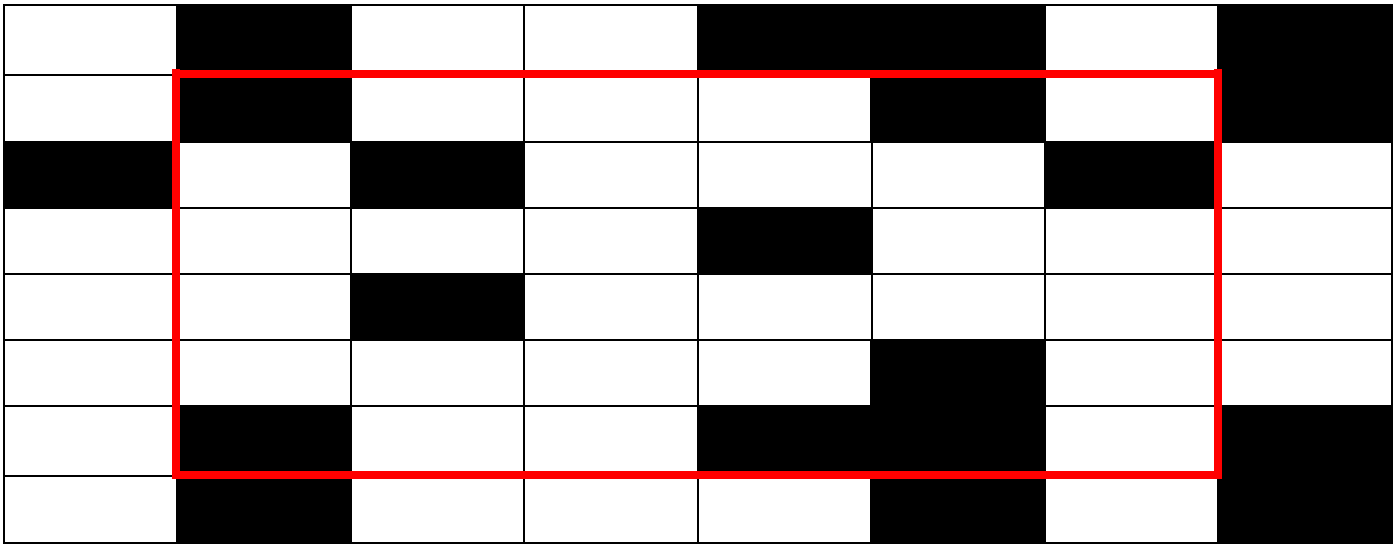




Apare o problema atunci cand celulele se gasesc la marginea sau coltul matricei, deoarece nu au toti cei 8 vecini. De aceea, starea automatului va fi vazuta ca un toroid. Pentru a simplifica problema, este suficient ca matricea initiala sa fie bordata cu o margine de o celula. Valorile din margine vor fi dupa cum urmeaza:

- in partea de sus se vor gasi valorile din ultima linie a matricei
- in partea de jos se vor gasi valorile din prima linie a matricei
- in partea din stanga se vor gasi valorile din ultima coloana a matricei
- in partea din dreapta se vor gasi valorile din prima coloana a matricei
- in colturi se vor gasi valorile din matrice aflate in colturile opuse.

Un exemplu puteti vedea mai jos, matricea originala fiind incadrata in chenarul rosu.



Regulile stabilite de Conway in Game of Life sunt urmatoarele:

- Daca o celula are mai putin de 2 vecini in viata (ALIVE) aceasta va fi moarta (DEAD) la iteratia urmatoare
- Daca o celula in viata (ALIVE) are 2 sau 3 vecini in viata, aceasta va fi in viata (ALIVE) la iteratia urmatoare
- Daca o celula are mai mult de 3 vecini in viata, aceasta va fi moarta (DEAD) la iteratia urmatoare
- Daca o celula moarta (DEAD) are 3 vecini vii, aceasta va fi in viata (ALIVE) la iteratia urmatoare

Pentru dimensiuni mici ale problemei, un astfel de automat, implementat serial, se executa pe un procesor modern intr-un timp rezonabil, dar cu cat dimensiunile cresc, cu atat trebuie utilizat procesorul eficient. De asemenea, pentru a trece de la o iteratie la alta se folosesc doua zone de memorie; in prima se retine configuratia la pasul i, iar in a doua configuratia la pasul i+1.

2. Cerinte

1) Implementati un program in C/C++ care rezolva problema Game of Life intr-o modalitate seriala. Acesta se va denumi `g_serial.c[pp]`. Binarul rezultat in urma compilarii se va denumi `g_serial` si va primi in linie de comanda urmatorii parametrii, in ordinea mentionata mai jos:
    a) Numele unui fisier ce contine starea initiala a problemei
    b) Numarul `N` de iteratii pe care trebuie sa le faca automatul
    c) Numele unui fisier in care se vor salva toate starile prin care trece automatul

    Formatul fisierului de intrare este urmatorul:
    a. Pe prima linie se va scrie numarul de linii (L) si coloane (C) a matricei, separate cu spatiu
    b. Pe urmatoarele L linii se vor gasi scrise separate printr-un spatiu starea fiecarei celule de pe linia respectiva astfel: pentru o celula in viata (ALIVE) se va scrie caracterul `'X'`, iar pentru o celula moarta (DEAD) se va scrie caracterul `'.'`

Fisierul de iesire va contine informatiile despre starea automatului la finalul tuturor iteratiilor, in formatul prezentat anterior. Nu este nevoie sa salvati numarul de linii si de coloane, ci doar starea finala.

Exemplu de rulare:
`./g_serial in.txt 100 out.txt`

Exemplu `in.txt`:

```
21 27
...........................
...........................
...........................
...........................
...........................
...........................
...........................
....X.........XX...........
....X..XXX....X.........XXX
....X............X.....XXX.
..................XX.......
....X.....................
.....X....................
...XXX....................
...........................
...........................
....X..X..................
........X..................
....X...X..................
.....XXXX..................
...........................
```

2) Paralelizati programul de la punctul 1, folosind OpenMP. Acesta se va denumi `g_omp.c[pp]`, iar binarul rezultat se va denumi `g_omp`. Parametrii primiti din linia de comanda, precum formatul datelor de iesire si modalitatea de salvare sunt aceeasi ca la punctul 1.

   Se vor folosi exact doua zone de memorie (doua matrici) pentru automat, iar la finalul fiecarei iteratii se va inlocui matricea din iteratia veche cu matricea din cea noua. Operatiunea de copiere dintre cele doua zone de memorie se va face tot in paralel.

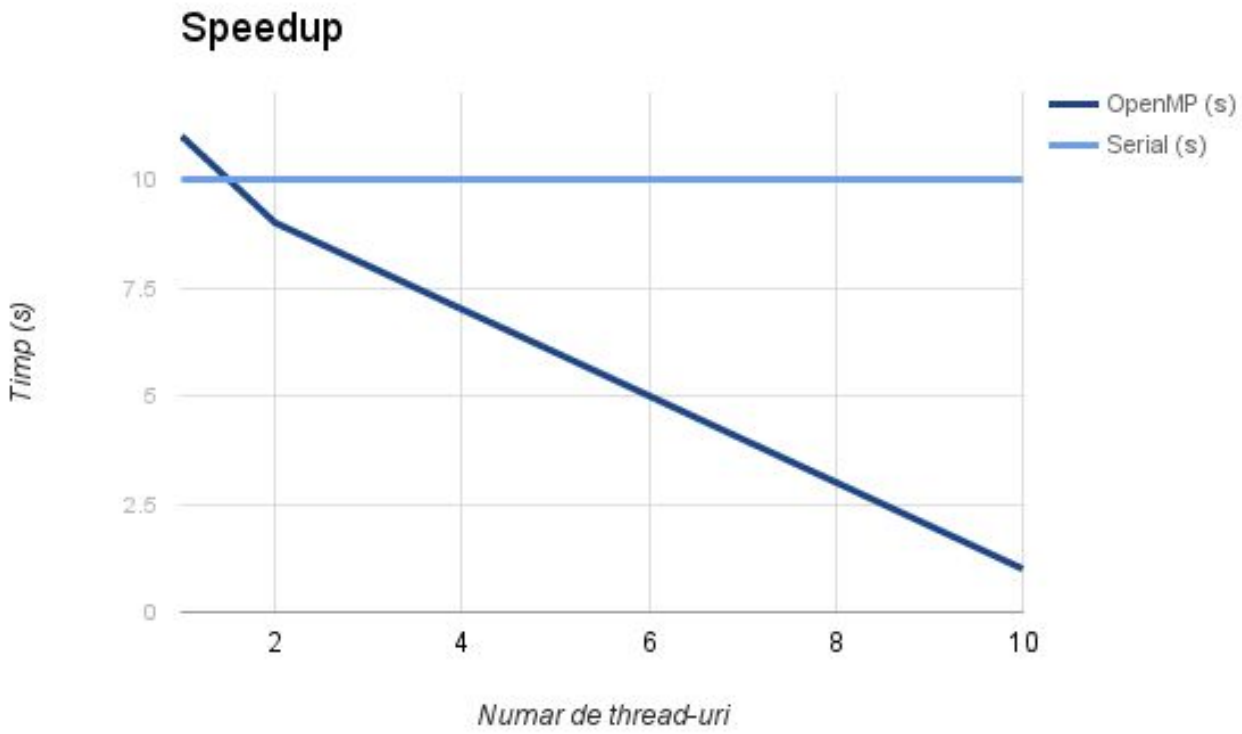
   Exemplu de rulare:
   `./g_omp in.txt 100 out.txt`

3) Rulati programul de la punctul 2 variind numarul de thread-uri. Veti alege 1, 2, 4 sau 8 thread-uri. Rularea la acest punct se va face pe FEP. Rezultatele obtinute la punctul 1 si 2 se vor scrie intr-un fisier denumit `results.[xls,xlsx,ods]`. De asemenea, in fisierul de rezultate, veti include un grafic in care sa reiasa speed-up-ul obtinut. Pe axa OY veti trece

timpul obtinut in urma rularii, iar pe axa OX veti trece numarul de thread-uri folosite.

ATENTIE! Veti fi depunctati daca implementarea paralela este ineficienta, comparativ cu varianta seriala.

De exemplu, graficul poate arata ca mai jos.



3. Predarea temei

Arhiva temei, in format **ZIP**, va contine:

- codul sursa pentru tema (`g_serial.c[pp]` si `g_omp.c[pp]`)
- un fisier `README` in care sa descrieti implementarea si modalitatea de paralelizare a problemei. Acesta poate fi in format txt, pdf, doc sau odt.
- un fisier `Makefile` reprezentand pasii de compilare. Trebuie sa existe urmatoarele reguli:
  - `clean` - sterge fisierele rezultate in urma compilarii
  - `serial` - compileaza `g_serial.c[pp]`
  - `parallel` - compileaza `g_omp.c[pp]` cu suport de OpenMP
- fisierul de rezultate.

Arhiva se va incarca pe site-ul de curs, in pagina aferenta temei.

4. Testarea temei

Testarea temei se va face pe FEP, folosind drept teste fisierele de intrare ce vor fi publicate pe site-ul de curs, plus un set de teste private.

5. Exemplu de rulare pe FEP

```
alecsandru@L:~$ scp g_omp.c alecsandru.patrascu@fep.grid.pub.ro
alecsandru@L:~$ ssh alecsandru.patrascu@fep.grid.pub.ro
[alecsandru.patrascu@fep7-1 ~]$ module load compilers/gnu-5.4.0
[alecsandru.patrascu@fep7-1 ~]$ make clean
[alecsandru.patrascu@fep7-1 ~]$ make parallel
[alecsandru.patrascu@fep7-1 ~]$ cat script.sh
#!/bin/bash
export OMP_NUM_THREADS=8
./g_omp $1 $2 $3

[alecsandru.patrascu@fep7-1~]$qsub   -cwd   -q   <QUEUE>   script.sh   in.txt   100
out.txt
```

- `<QUEUE>` - coada de procesare: `ibm-nehalem.q` sau `campus-haswell.q`
- `script.sh` - scriptul care se va rula pe cluster
- `in.txt` - starea initiala a problemei
- `100` - numarul de iteratii
- `out.txt` - starea finala a problemei

```
[alecsandru.patrascu@fep7-1 ~]$ qstat
job-ID prior name user state submit/start at queue slots ja-task-ID
-----------------------------------------------------------------
532370 0.00000 script.sh alecsandru.patracu qw 11/10/2016 23:49:19 4
```

Output-ul îl găsiți în fișierul `script.sh.o532370` (este numele fișierului concatenat cu .o și ID-ul job-ului) după ce job-ul nu mai apare la comanda `qstat`.