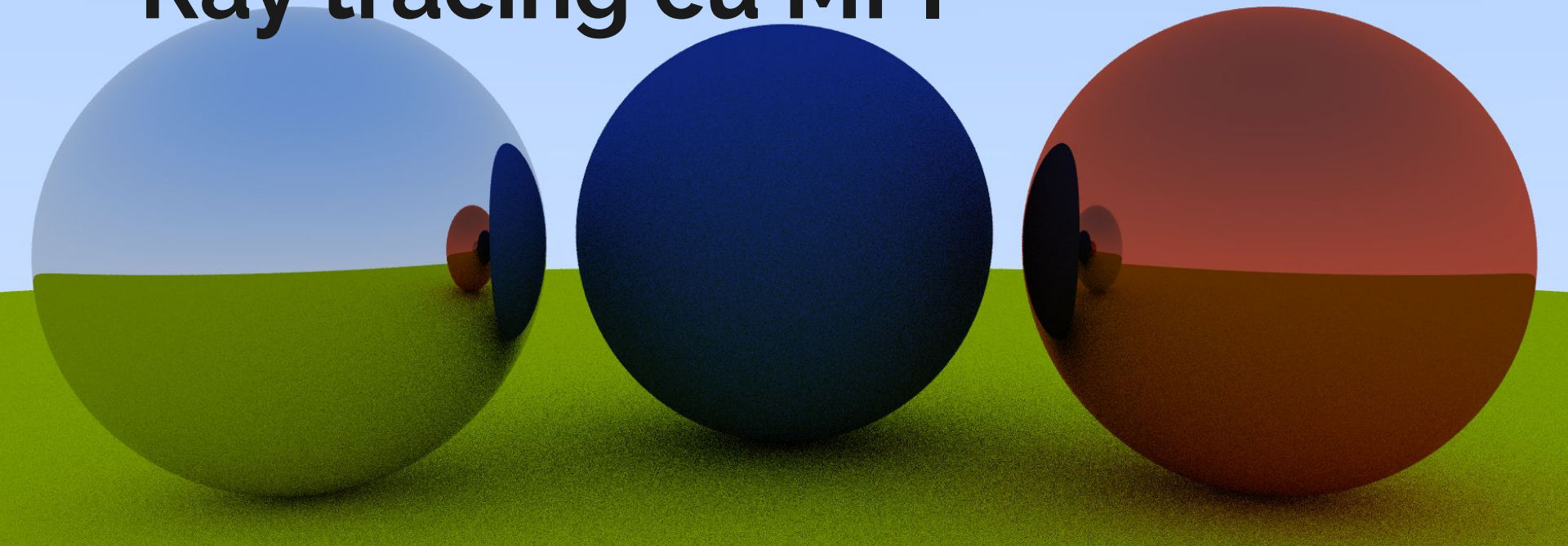
Mihai-George Licu
Radu-Andrei Moraru
Gr. 462

Ray tracing cu MPI



Outline

Definitia ray-tracing

Metoda de producere de imagini cu potential mare de paralelizare.

Design paralel

Impartirea load-ului intre procesoare in blocuri contiguous/noncontiguous.

Influenta topologiei asupra solutiei paralele

Compararea topologiilor de tip 2d grid/hipercub

Ce este ray-tracing?

- Tehnică de generare a imaginilor din scene 3D prin simularea traiectoriilor razelor de lumină
- Pentru fiecare pixel din imagine se traseaza o raza
- Culoarea si traiectoria razei sunt afectate de obiectele de care se loveste
- Culoarea rezultata a razei este culoarea luata de pixel

De ce este nevoie de paralelizare?

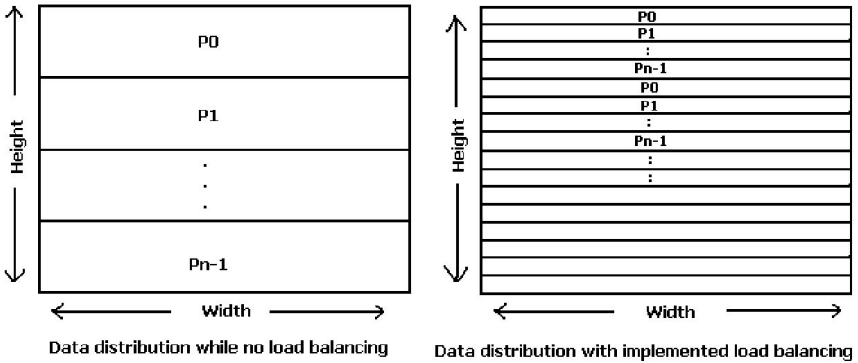
- Pentru imagini de înaltă rezoluție, numărul de raze de lumină de calculat este imens
- Razele sunt independente, pot fi procesate in paralel
- Paralelizarea permite accelerarea semnificativă a timpului de randare

Design paralel - Load balancing

Generarea imaginii se poate paraleliza divizand imaginea in blocuri dupa randuri, coloane, sau ambele.

Dorim sa ne asiguram ca munca este relativ echilibrat distribuita intre procesoare.

Load format din blocuri nonadiacente => sanse mai mari ca diferentele intre load-uri sa fie mici



Influența topologiei asupra soluției paralele

Topologia influențează modul în care procesele comunică și interacționează între ele.

Principale considerari:

- Dorim sa scriem imaginea pe un singur disk
- Colectarea rezultatelor fiecarui procesor poate fi facuta cu o operatie de tip Gather
- Operatia de gather nu difera in complexitate in cazul topologiilor de tip ring/grid/hypercube.

Complexitatile topologiilor considerate pentru implementare:

- Ring: Gather in O(p)
- Grid: Gather in O(p)

Specificatia solutiei

- Scena 3D este formata din culoarea cerului (gradient albastru) si sfere de diferite dimensiuni, pozitii, si materiale.
- Tipurile de materiale alese:
  - Lambertian (random diffusion)
  - Metal (reflexie perfecta)
- Imaginea este generata intr-un fisier PPM pentru simplitate.
- Pentru anti-aliasing, calculam media mai multor raze trimise prin pixel.

Mediu de dezvoltare si cerinte de sistem

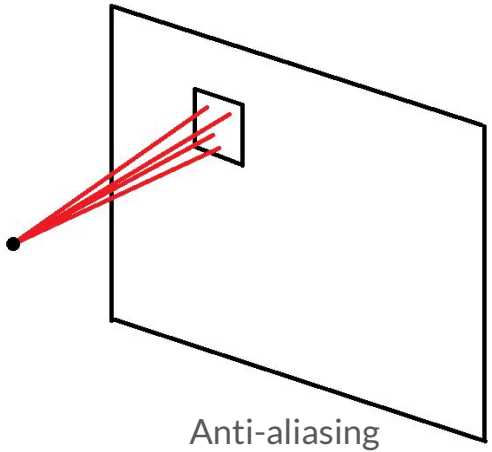
- Proiect realizat cu OpenMPI pe sisteme de operare bazate pe Linux.
- Pentru a genera o imagine de rezolutie 1920x1152 in sub 1s am folosit un procesor i5 13th gen.
- Necesarul total de memorie pentru o imagine cu N pixeli: 2*N*3*sizeof(double)

Design

- Definirea scenei ca un set de obiecte cu pozitii, dimensiuni, si materiale diferite.
- Calcularea culorii unui pixel fiind date doar pozitia lui in imagine si scena.
- Impartirea load-ului intre procesoare:
  - Fara load balancing: procesoarele iau blocuri contiguous de dimensiuni egale din imagine
  - Cu load balancing: Procesorul i din N ia randurile de pe pozitiile i+N*k pentru k variabil
- Reconstruirea si salvearea imaginii:
  - Efectuata doar de procesul 0
  - Procesul depinde de metoda de impartire a load-ului

Implementare - Ray Tracing

- Imaginea este considerata un plan de numere reale (Viewport)
- Fiecare pixel este considerat un patrat in plan
- Razele sunt trimise dintr-un punct aflat in spatele viewport-ului prin pixeli.
- Pentru fiecare pixel sunt trase mai multe raze prin puncte alese aleator (anti-aliasing)
- Se gaseste cel mai apropiat obiect cu care se intersecteaza raza. Se va relua procesul pentru o raza noua.
- In functie de material, obiectul modifica culoarea si traiectoria urmatoarei raze



Anti-aliasing

Implementare - paralelizare

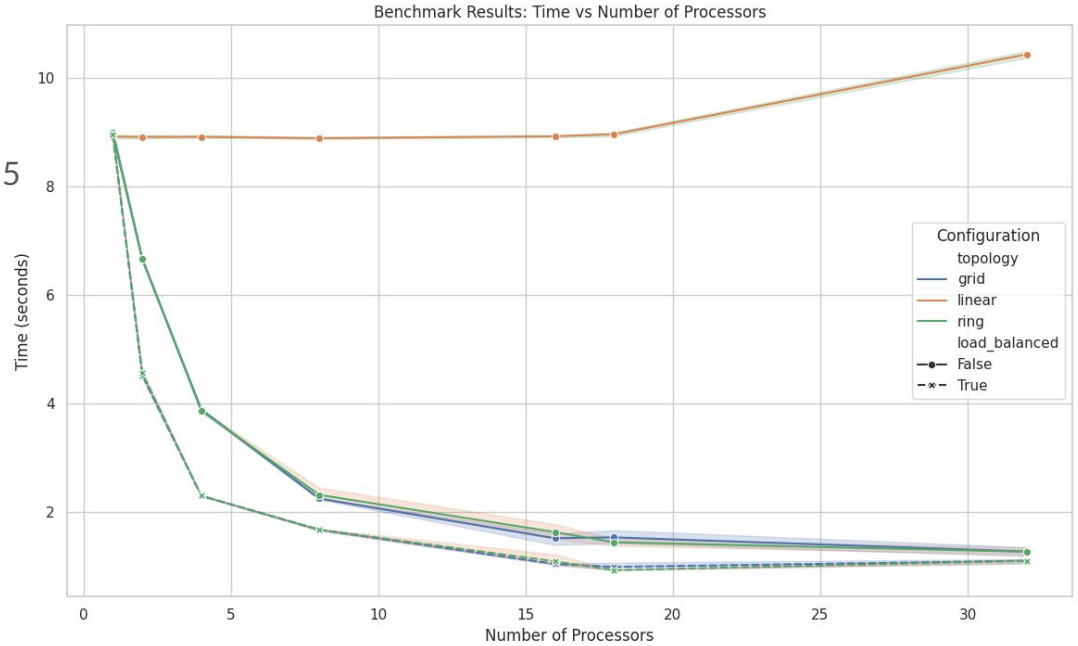
- Scena si dimensiunile imaginii sunt hard-coded si astfel incarcate de toate procesoarele.
- Se creeaza un comunicator pentru topologia aleasa:
  - Ring: comunicator cartezian cu o singura dimensiune si wrap-around
  - Grid: comunicator cartezian bidimensional
- Fiecare procesor isi aloca imageHeight/numWorkers * imageWidth * sizeof(pixel) pentru rezultatele locale
- Procesorul i calculeaza culoarea pixelilor de pe liniile alocate (diferite daca se foloseste load balancing)
- Rezultatele sunt colectate intr-un buffer mentinut doar de procesorul cu rang 0.
- Procesorul cu rang 0 reconstruieste imaginea.
  - Pentru solutia fara load balancing nu este nevoie de reordonare.
  - Pentru solutia cu load balancing se reordoneaza liniile.

Experimente

- Rulate pe un procesor cu 12 cores (i5 13th gen). Nu au fost folosite arhitecturi distribuite.

  Accelerare
  Sp(n) = t1(n) / tp(n)
  Sp(16) = 9/1.8 = 5

  Eficienta
  E = Sp/p = 5/16 = 0.3125



Concluzii

- Topologia nu a avut un efect considerabil in timpul rularii. (numarul de procesoare utilizate in experimente este redus)
- Metoda de load balancing utilizata a adus imbunatatiri considerabile la timpul de rulare.
- O arhitectura SIMD/SIMT ar avea rezultate considerabil mai bune