Parallel Diffusion Limited Aggregation:

Parallel and Distributed Programming

Team 02
Phan Manh Tuan - 20194461
Le Truong Giang - 20194430
Nguyen Duc Phu - 20194420
Nguyen Minh Chau - 20194420

01 Algorithm
DLA Algorithm

02 Main Function
Main function of C code using MPI

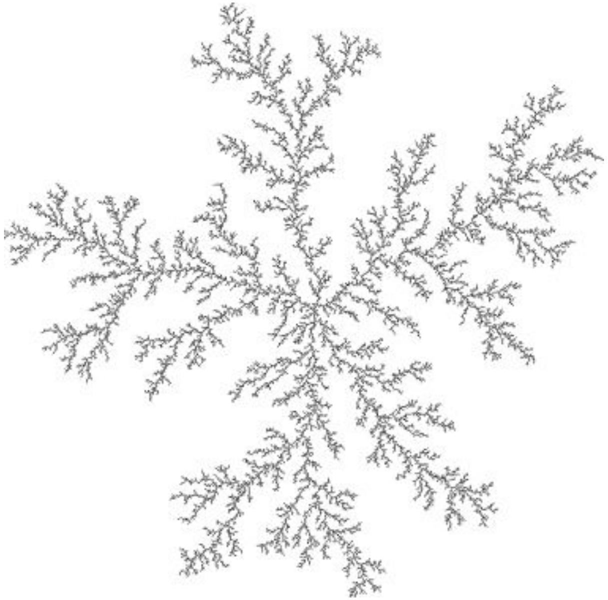
03 Initialization
How we initialize the data

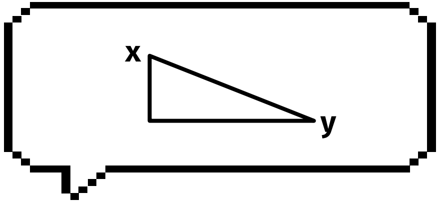
04 SOR Iteration
Use Successive Over Relaxation method to solve Laplace Equation

05 Growth
Grow the Object

DLA

Diffusion Limited Aggregation (DLA) is a model for non-equilibrium growth, where growth is determined by diffusing particles





01

Algorithm

DLA Algorithm

Basic Algorithm

- Solve Laplace equation to get distribution of nutrients, assume that the object is a sink (i.e. c = 0 on the object)
- Let the object grow
- Go back to first step

The first step in above algorithm is done by a parallel SOR iteration

Simulation

Boundary Condition

X-direction and
Y-direction

Object Initialization

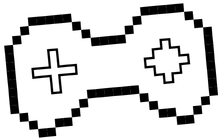
Where Object start
to grow

Hyperparams

Parameter in
experiment

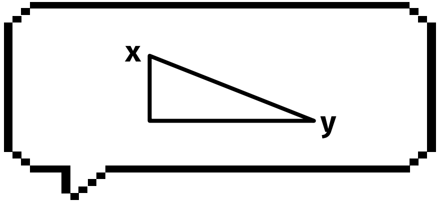
Boundary Condition and Object Init

- Periodic boundary conditions in x-direction
- Fixed value for the upper and lower boundary in y-direction
- The object start at the bottom center of the grid



Hyperparams

	Value
Number of Iteration	800 & 2000
Size of grid	200 x 200
Tolerance value for convergence	0.001
Omega	1.9
Upper and Lower boundary	C0 = 1 cN = 0



02

Main Function

Main Function in C using MPI

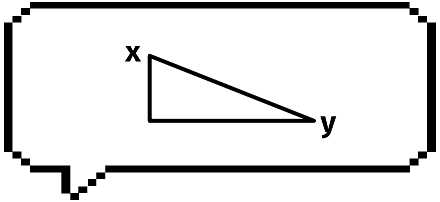
```c
    KhoiTao(C_old, C_current, O, candidates);

    for (k = 0; k < iter; k++)
    {
        SOR();
        for (i = 0; i < Np; ++i)
            for (j = 0; j < N; ++j)
                *(C_old + i * N + j) = *(C_current + i * N + j);
        growth();
    }

    float *C_current_global;
    int *O_global;
    // print
    if (rank == 0)
    {
        C_current_global = (float *)malloc(N * N * sizeof(float));
        O_global = (int *)malloc(N * N * sizeof(int));
    }
    MPI_Gather(C_current, Np * N, MPI_FLOAT, C_current_global, Np * N, MPI_FLOAT, 0, MPI_COMM_WORLD);
    MPI_Gather(O, Np * N, MPI_INT, O_global, Np * N, MPI_INT, 0, MPI_COMM_WORLD);
```

Initialization

SOR Iteration

Grow Step

Gather to Root



03

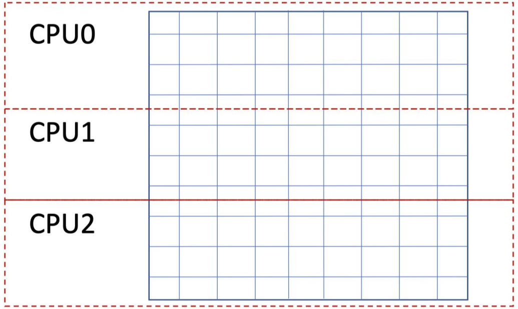
Initialization

How we initialize the data

Initialization

- Each CPU : Initialize independently

→ No need to distribute Input data from Root to all other CPUs




```
    // C  Initialization
    for (i = 0; i < Np; ++i)
        for (j = 0; j < N; ++j)
        {
            *(C_current + i * N + j) = 0;
            *(C_old + i * N + j) = 0;
            *(O + i * N + j) = 0;
        }

    // Object Initialization
    if (rank == size - 1)
        *(O + (Np - 1) * N + N / 2) = 1;
```

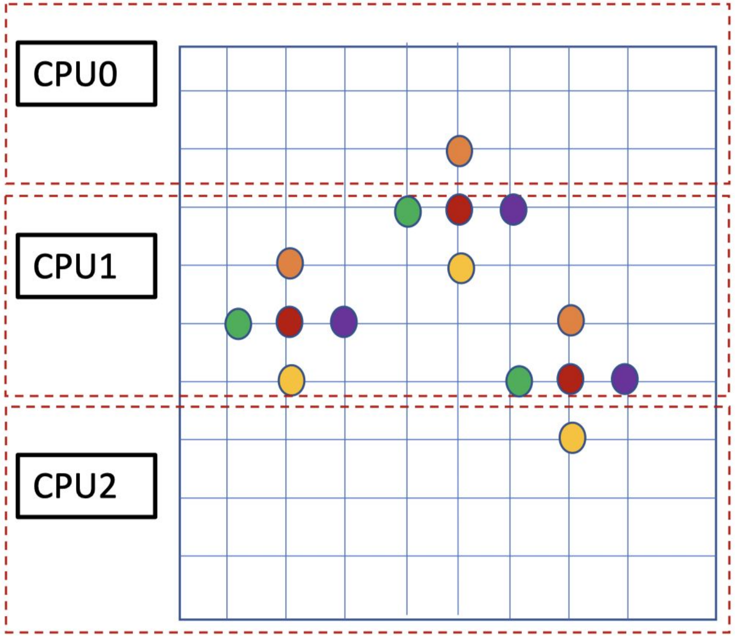
04

SOR Iteration

Use Successive Over Relaxation method to solve Laplace Equation

Communication

- Color the computational grid as a checkerboard, with red and black grid points.
- First update all red points and next update all black points
- Communicate **C_above** and **C_below**

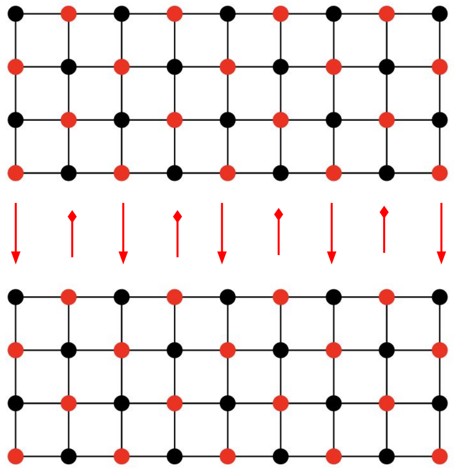


Communication

- Color the computational grid as a checkerboard, with red and black grid points.
- Communicate **C_above** and **C_below**

CPU (n+1) send up data to **C_above** of CPU (n)

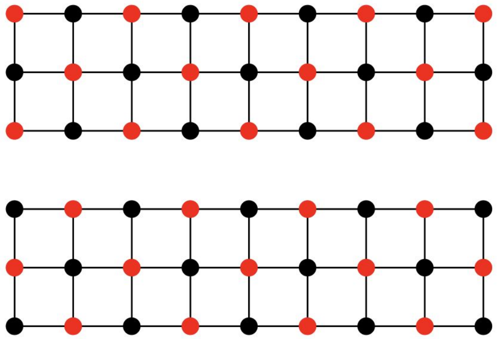
CPU (n) send down data to **C_below** of CPU (n)



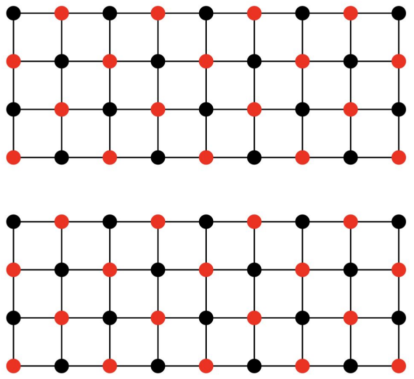
Red Turn

Communication

```c
// He so cho tung Np khac nhau
dieu_chinh = (Np * rank + 1 + r) % 2;

// Send down
if (rank != size - 1)
    MPI_Send(C_current + (Np - 1) * N + (dieu_chinh + Np) % 2, 1, everytwice, rank + 1, 0, MPI_COMM_WORLD);

// Send up
if (rank != 0)
    MPI_Send(C_current + 1 - dieu_chinh, 1, everytwice, rank - 1, 1, MPI_COMM_WORLD);

// Receive from below
if (rank != size - 1)
    MPI_Recv(C_below, (N + 1) / 2, MPI_FLOAT, rank + 1, 1, MPI_COMM_WORLD, &status);
else
{
    for (i = 0; i < (N + 1) / 2; ++i)
        C_below[i] = cN;
}

// Receive from above
if (rank != 0)
    MPI_Recv(C_above, (N + 1) / 2, MPI_FLOAT, rank - 1, 0, MPI_COMM_WORLD, &status);
else
{
    for (i = 0; i < (N + 1) / 2; ++i)
        C_above[i] = c0;
}
```



Odd Np



Even Np

Calculation

```c
  // ignore object position
  if (*(O + i * N + j) == 1)
      continue;

  // ignore half of the grid
  if ((rank * Np + i + j) % 2 == r)
      continue;

  left = (j == 0) ? *(C_current + i * N + (N - 1)) : *(C_current + i * N + j - 1);
  right = (j == N - 1) ? *(C_current + i * N) : *(C_current + i * N + j + 1);
  up = (i == 0) ? *(C_above + j / 2) : *(C_current + (i - 1) * N + j);
  down = (i == Np - 1) ? *(C_below + j / 2) : *(C_current + (i + 1) * N + j);

  *(C_current + i * N + j) = (1 - omega) * *(C_current + i * N + j) + omega * (left + right + up + down) / 4;

  *alpha = fmax(*alpha, fabs(*(C_current + i * N + j) - *(C_old + i * N + j)));
```

Communication

```
      MPI_Allreduce(alpha, &global_alpha, 1, MPI_FLOAT, MPI_MAX, MPI_COMM_WORLD);
  } while (global_alpha > tol);
```

```
  MPI_Datatype everytwice;
  MPI_Type_vector((N + 1) / 2, 1, 2, MPI_FLOAT, &everytwice);
  MPI_Type_commit(&everytwice);
```

05

Growth

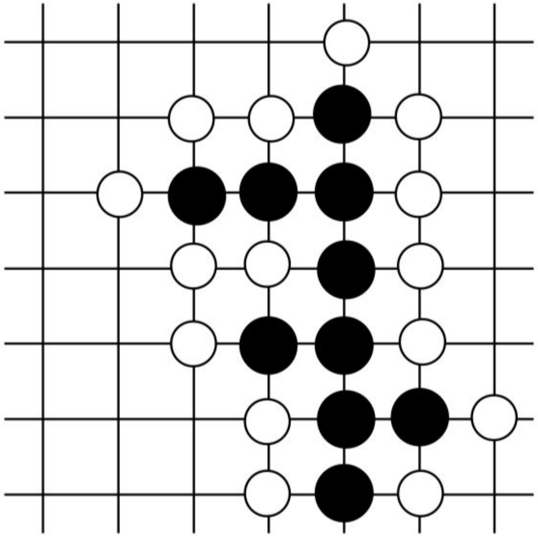
Grow the Object

Growth

- Growing object requires 3 steps:
  - Determine growth candidates
  - Determine growth probabilities
  - Grow

Calculation of the growth probabilities requires a global communication

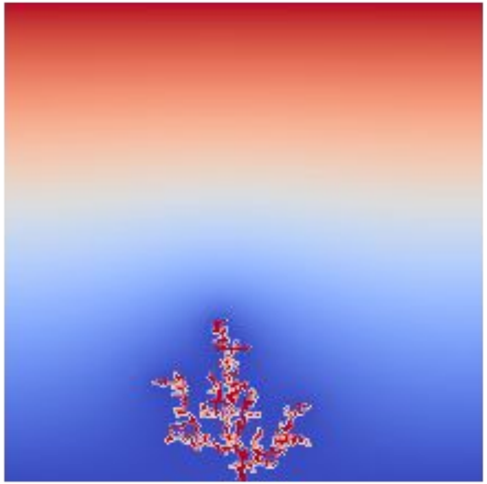
$$p_g\left((i,j)\in\circ\rightarrow(i,j)\in\bullet\right)=\frac{\left(c_{i,j}\right)^{\eta}}{\sum\limits_{(i,j)\in\circ}\left(c_{i,j}\right)^{\eta}}.$$



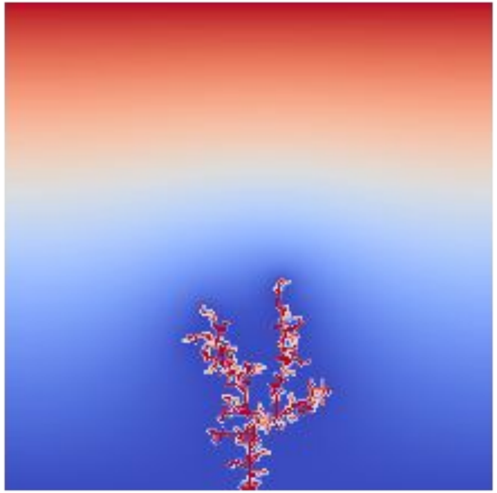
Growth

```c
    // calculate candidates and nutri
    for (i = 0; i < Np; i++)
    {
        for (j = 0; j < N; j++)
        {
            if (*(O + i * N + j) == 1)
            {
                continue;
            }

            left = (j == 0) ? 0 : *(O + i * N + j - 1);
            right = (j == N - 1) ? 0 : *(O + i * N + j + 1);
            up = (i == 0) ? *(O_above + j) : *(O + (i - 1) * N + j);
            down = (i == Np - 1) ? *(O_below + j) : *(O + (i + 1) * N + j);

            if ((left == 1 || right == 1 || up == 1 || down == 1))
            {
                *(candidates + i * N + j) = 1;
                *nutri += *(C_current + i * N + j);
            }
            else
                *(candidates + i * N + j) = 0;
        }
    }

    // Reduce
    MPI_Allreduce(nutri, &global_nutri, 1, MPI_FLOAT, MPI_SUM, MPI_COMM_WORLD);
```

Some Results

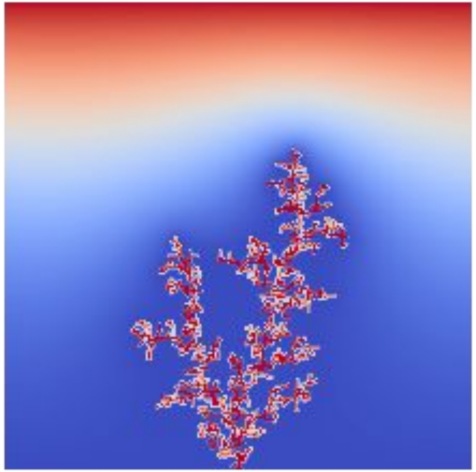


800 step serial

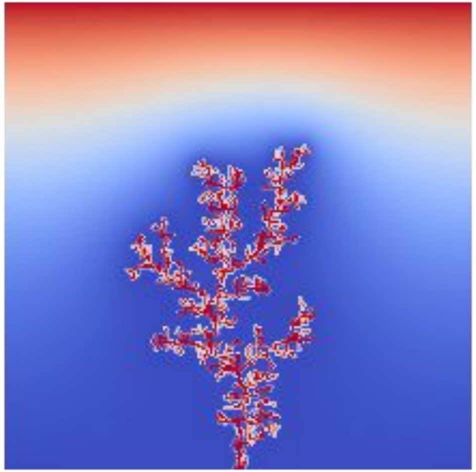


800 step parallel

Some Results



2000 step serial



2000 step parallel

THE END!

Thank you for listening