Programming assignment #1

Multicore systems

Sugar scape model

- Artificially intelligent agent-based social simulation
  - The original model presented by J.Epstein & R.Axtell
- Universe (Modified version)
  - n x n, 2D orthogonal grid of square patches
  - Patch has some sugar
  - Some of the people live in this grid
  - Each time step, people lives or dies depending on some actions
- Actions
  - 1. People move one direction depending on 4 neighbor patch's amount of sugar
  - 2. Some people merged in same patches
  - 2. People eat sugar in the patch
  - 3. People consumes some sugar.
  - 4. If the sugar is shortage, people will die.

3D Sugar scape model

- Cubic patches in 3D universe
  - One person will has 6 compass direction

- Actions
  - 1. People move one direction **depending on 6 neighbor patch's amount of sugar**
  - 2. Some people merged in same patches
  - 3. People eat sugar in the patch
  - 4. People consumes some sugar.
  - 5. If the sugar is shortage, people will die.
  - 6. Generate people and sugar in the 3D universe as much as dead or consumed

Programming

- 3D Sugar scape model in two programming models
  - Sequential programming
  - Parallel Programming with Pthreads , OpenMP, MPI
- Input file
  - Parameters about sugar scape model, random seeds, steps to run
- Output
  - Measured time
    - Report on log file in microseconds (using gettimeofday() function)
    - Core work of simulation excluding file read/write, initialization
    - statistic of each stage.
  - Output file
    - Resulting universe of sugar maps, people information.

Initial attribute

- People
  - (x, y, z) axis
  - Eat : The amount of sugar that eaten at one time
    - #Eat_min <= #Eat <= #Eat_max
  - Consume : The amount of consumed sugar that each turn
    - #Consume_min <= #Consume <= #Consume_max
  - Sugar : The amount of holding sugar
    - #Sugar_min <= #Sugar <= #Sugar_max
- Map
  - Size of map
  - Sugar : The amount of sugar that initialized each patch
    - #Map_Sugar_min <= #Sugar <= #Map_Sugar_max

Input file(1/3)

- Parameters about sugar scape model
  - total_loop
  - map_size
  - total_people
  - min_map_sugar
  - max_map_sugar
  - min_eating
  - max_eating
  - min_consume
  - max_consume
  - min_sugar
  - max_sugar

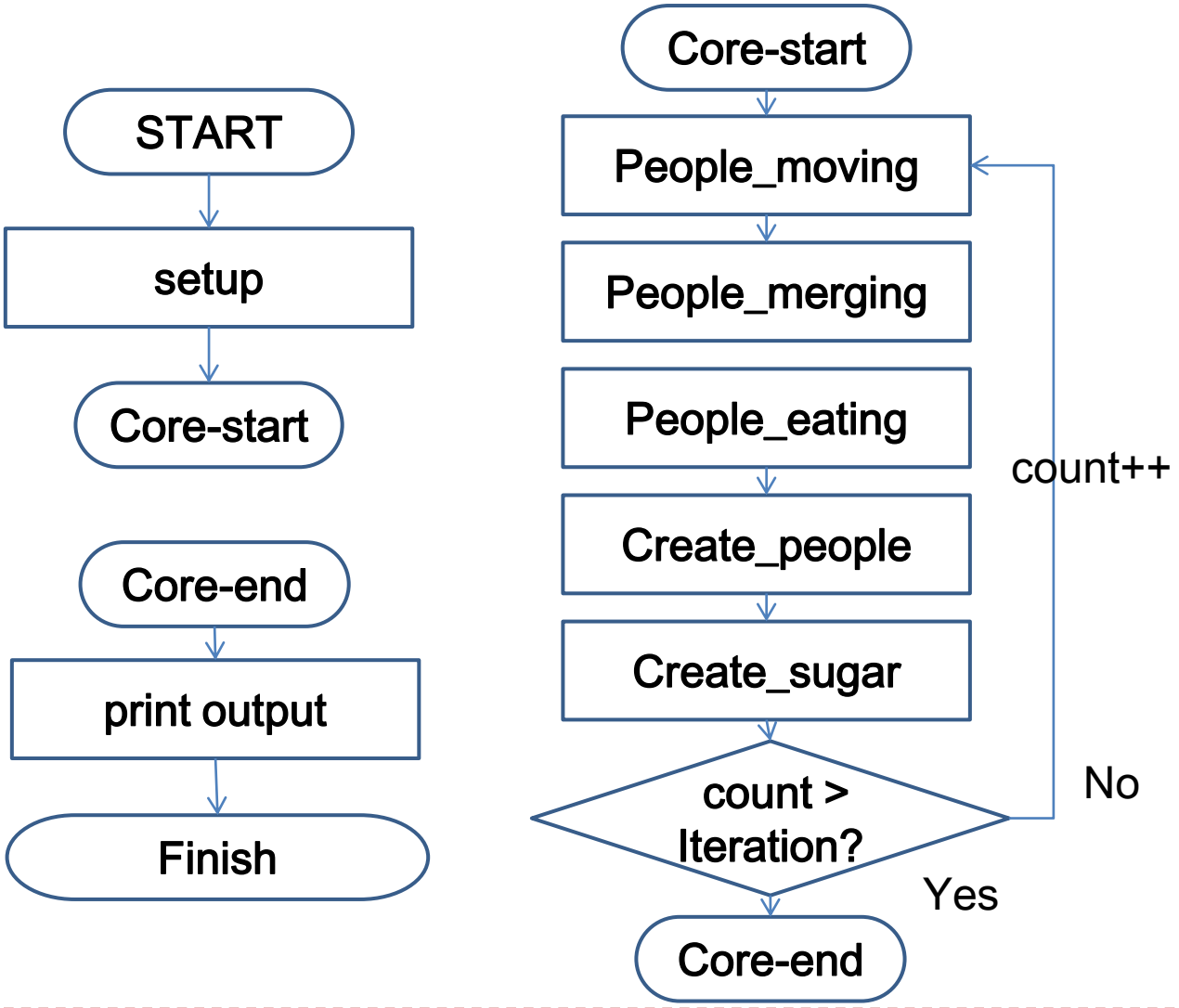
Input file(2/3)

- Random seeds about sugar map
  - random_seed_map_x
  - random_seed_map_y
  - random_seed_map_z
  - random_seed_map_sugar

Input file(3/3)

- Random seeds about people
  - random_seed_people_x
  - random_seed_people_y
  - random_seed_people_z
  - random_seed_people_eat
  - random_seed_people_consume
  - random_seed_people_sugar

Flow chart


START
setup
Core-start
Core-end
print output
Finish
Core-start
People_moving
People_merging
People_eating
Create_people
Create_sugar
count++
count > Iteration?
No
Yes
Core-end

Setup

- Setting parameter
  - Using input file
- Generate files
  - Sugar map file
  - People information file
- Read generated files
  - Make a Data structures about that.

People moving

- Look in 6 compass directions
  - (+x, +y, +z, -x, -y, -z)
- Select patches with the highest sugar value
- If several patches exist with the same value, choose the patch on the basis of the priority
  - Direction priority : +x, +y, +z, -x, -y, -z

People merging

- If several people exist in the same patch, merge the people.

- Merge rule
  - eat = floor(average(people->eat))
  - consume = floor(average(people->consume))
  - sugar = floor(average(people->sugar))

- Count
  - People dead count
  - Sugar consume count

People eating

- Eat sugar in the patch.
- Make people consume sugar

- Eating Rules
  - Rule #1
    - Amount of eating sugar = min(patch's sugar, people's eat)
  - Rule #2
    - Sugar += pre(people's sugar) + min(patch's sugar, people's eat)
  - Rule #3
    - Sugar – consume > 0 : live
    - Sugar – consume <= 0 : dead

- Count
  - People dead count
  - Sugar consume count

Create people

- Create people as much as dead people count

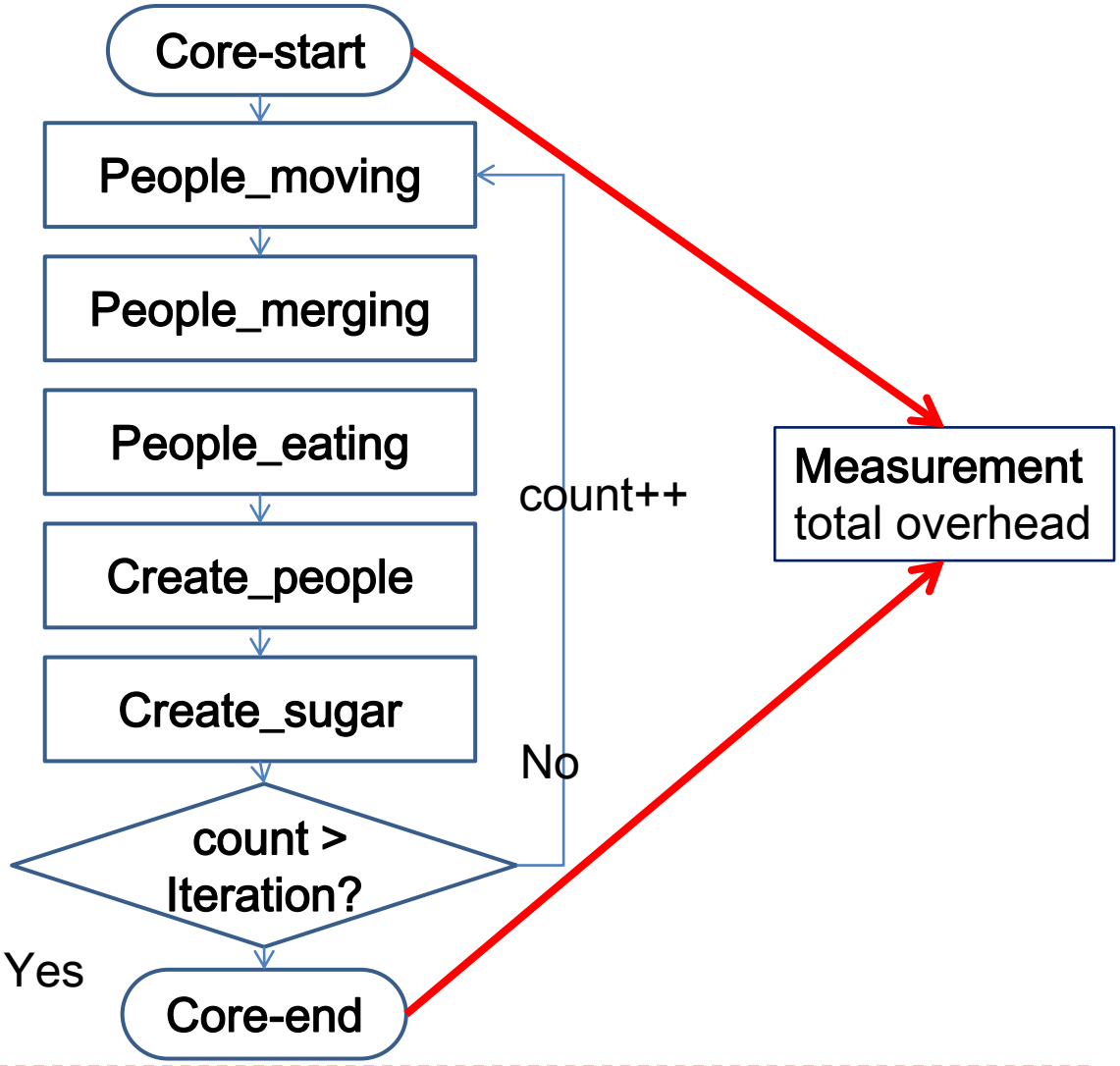
- Create Rule
  - Random number should be generated using that
    - **x_axis** : random_seed_people_x
    - **y_axis** : random_seed_people_y
    - **z_axis** : random_seed_people_z
    - **eat** : random_seed_people_eat
    - **consume** : random_seed_people_consume
    - **sugar** : random_seed_people_sugar

Create sugar

- Create sugar as much as consumed sugar count

- Create Rule
  - Random number should be generated using that
    - **x_axis** : random_seed_map_x
    - **y_axis** : random_seed_map_y
    - **z_axis** : random_seed_map_z
  - Each sugar produced by one in the patch

Measurement time



Submission

- Submit through “icampus

- Due date
  - seq : 5/1
  - pthread : 5/11
  - openmp : 5/21
  - mpi : 5/31

- studentID.tar
  - Makefile
  - Your algorithm PPT
  - README

Additional items

- T.A.: Byeonghun Hyeon (#85557)
  - gusqudgnskku@gmail.com
  - Test the performance to measure your parallelization effort
  - Test the each stage's parallelization effort

- Penalty
  - Delayed submission: -10% per day

- Score
  - Right answer : 50%
  - Speed up : X = (my_parall_time / fast_seq_time)
    my_score % = (X / max_speed_up) * 50%
  - pthrad, openmp, mpi == (1 : 1 : 1) == ((0.5:0.5), (0.5:0.5), (0.5:0.5))