Outline

- Brief History
- Introduction to Cellular Automata
- Cellular Automata in Parallel Computing
- Common Applications
- Programming Environments
- Questions
Brief History

- 1947 - John von Neumann developed the idea for cellular automata to solve the problem of self-replicating systems.
  - His original purpose was to build a self-replicating robot.
  - He developed the idea along with Polish mathematician Stanislaw Ulam who studied crystal growth.
- 1969 - Konrad Zuse from Germany published a book titled *Calculating Space* which proposed the entire universe was the result of a giant cellular automata system.
- 1970 - The idea of cellular automata was popularized by Jon Conway and The Game of Life.
- 1983 - Stephen Wolfram published a series of papers investigating elementary cellular automata. This led to the idea of its application of modeling natural systems.
Introduction

- Cellular Automata is a discrete dynamic system.
- The system consists of a finite number of cells.
- The cells are arranged in a grid or lattice.
- Each cell has neighboring cells. The number of which depends on the arrangement of the cells.
Cell Arrangements

- Common cell arrangements are 2-D and 3-D grids
Cell Neighbors

- Cell lattices can be arranged in any geometrically possible shapes, allowing different neighbors
Cellular System

- Each cell has a finite number of states.
- The system has a global behavior that defines how cells react with their neighbors.
- At each time step, this behavior will be applied to the cell based on the states of its neighbors, causing that cell to transition to a different state.
- Once all cells transition to that new state, a new generation has been formed.
Cellular System

- Each cell has a starting state.
- Commonly, all cells have the same starting state except a very few that differ.
- This arrangement is referred to as the configuration of the system.
Simple Cellular Automata

**Rule table φ:**

<table>
<thead>
<tr>
<th>neighborhood η:</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>output bit:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**Lattice:**

<table>
<thead>
<tr>
<th>Neighborhood η</th>
<th>r = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>t = 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>t = 1</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0 0 1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1 1 1 1 0 1 1</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></td>
</tr>
</tbody>
</table>
Simple Cellular Automata

A cellular system has a 1-D arrangement and the following global behaviors

The system starts with one black square in the middle

As the system evolves, obvious patterns are formed
Example

QuickTime™ and a MPEG-4 Video decompressor are needed to see this picture.
Cellular Automata and Parallel Systems

- Cellular Automata is an ideal model for parallel processing.
- Cells are easily mapped to processors in groups.
- Communication is minimum since cells only talk with cells in their neighborhood.
- Global behaviors allows for a SIMD system.
Parallel Computing Issues

- One issue with the MIMD model is dealing with cells around the borderer.
  - Boarding cells have neighborhoods of different sizes than non-boarder cells.
- These cells can either have different behaviors or their neighborhoods can be wrapped around the grid to form a torus.
Parallel Computing Issues

- Problems such as image processing can have large groups of idle cells. This could leave a parallel system heavily unbalanced and inefficient.
  - Problems like this can sometimes be load balanced by dynamically assigning active cells to processors with smaller loads.
Common Applications

- Game of Life
- Fractal Patterns
- Modeling Chaotic Behavior
- Cryptography
- Solving Wave Systems, Fluid Patterns, and Fluid Turbulences
- Modeling Circuit Systems
- Simulating Forest Fires
- Image Processing
Image Processing

- Image Skeletonization
- The subject of an image is continually thinned until only a skeleton remains.
- Used to study movement and understand illegible text

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.
Image Processing

- Image processing is an ideal problem for a dynamically load balanced parallel cellular automata system.

QuickTime™ and a TIFF (LZW) decompressor are needed to see this picture.
Fluid Dynamics

- Fluid patterns are often modeled by differential equations that vaguely resemble the original system.
- These patterns can be modeled easily in cellular automata systems by having cells represent each particle in the system.
- The particles then behave based on elementary physical laws.
Fluid Dynamics Example

- Rules
  - A molecule with empty space can move forward to a random position
  - A molecule with behind a cell already filled with molecules can move randomly in its own row
  - A molecule behind a barrier can move randomly in its own row
Fluid Dynamics Example

- The results are fluid-like motion
Route Planning

- Cellular Automata has been used to create parallelizable route planners
- Route planning factors
  - Weather
  - Visibility
  - Altitude
  - Time
  - Fuel
  - Safety
Parallelizable Route Planner
CAMELot

- An environment for the programming and seamlessly parallel execution of Cellular Automata
- Supports CARPET language
- Programming environment and GUI
- Visualization of Simulated Data
- Built on MPICH2, an implementation of MPI
- MPICH adds portability to MPI
dimension 2;
radius 1;
state ( unsigned char life; );
neighbour Moore[] ( [ 0,-1], [-1,-1], [-1, 0], [-1, 1],
                  [ 0, 1], [ 1, 1], [ 1, 0], [ 1,-1] );
deterministic;

register int i;
register int sum;
{
  sum = 0;
  for( i=0 ; i<8; i++)
    sum = Moore[i].life + sum;
  if ( sum == 3 || ( sum == 2 && cell_life == 1) )
    update (cell_life, alive);
  else
    update (cell_life, dead);
}
dimension 2;
radius 1;
state ( unsigned char
neighbour Moore[] ( [ ]
  deterministic;

})
register int i;
register int sum;
{
  sum = 0;
  for( i=0 ; i<8; i++)
    sum = Moore[i]_life + sum;

  if ( sum = 3 || ( sum = 2 && cell_life = 1))
    update (cell_life, alive);
  else
    update (cell_life, dead);
}
CARPET

- CellulAR Programming Environment
- CARPET implements cellular automata in a parallel processing environment
- Allows for parallel cellular systems by abstracting the cellular automation from the parallel architecture
- High level language based on C with additional constructs for cellular automata
CARPET Programming

- Declarations for dimensions of automation, radius of neighborhood, pattern of neighborhood, and state of cells

\[ \text{state( int particles, float temperature)} \]

Neighbor Neuman[4]( [0,-1]North,[-1,0]West,[0,1]South, [1,0]East);
Neighbor Moore[8]( [1,-1]Neast, [0,-1]North, [-1,-1]Nwest,
[-1,0]West, [1,0]East, [-1,1]Swest, [0,1]South, [1,1]SEast)
Forest Fire Simulation with CARPET

- The fire starts at a specific point and spreads to neighboring trees.
- Fire spreads in four directions, N, S, E, W.
- Each cell is either Trees, Fire, Land, or Dead.

![Simulation images]
Forest Fire Simulation with CARPET

```c
#define tree 2
#define fire 0
#define dead 3
#define land 1

cadef {
    dimension 2;
    radius 1;
    state (short ground);
    neighbor moore[8] ( [0,-1]North,[-1,-1]NorthWest, [-1,0]West, [-1,1]SouthWest,[0,1]South, [1,1]SouthEast, ….
    parameter (dens 0.6);
}

float px;
{
    if (step == 0) {
        px = ((float) rand())/RAND_MAX;
        if (px < dens)
            update(cell_ground, tree);
        else
            update(cell_ground, land);
    }
    else
        if((cell_ground == tree) && (North_ground == fire || South_ground == fire || East_ground == fire ||
             West_ground == fire || NorthWest_ground == fire || SouthWest_ground == fire ||
             SouthEast_ground == fire || NorthEast_ground == fire))
            update(cell_ground, fire);
        else
            if (cell_ground == fire)
                update(cell_ground, dead);
    }
```

#include space 0
#define wire 1
#define electhead 2
#define electail 3
cadef
{
dimension 2; /*bidimensional lattice */
radius 1;
state (short content);
    neighbor moore[8] ([0,-1]North, [-1,-1]NorthWest, [-1,0]West,
    [-1,1]SouthWest, [0,1]South, [1,1] SouthEast,
    [1,0]East, [1,-1]NorthEast);
}
int i; short count;
{
count = 0;
for (i = 0; i<8; i++)
    if (moore[i]_content == electhead)
        count = count + 1;
switch (cell_content)
{
case electail : update(cell_content, wire); break;
case electhead: update(cell_content, electail); break;
case wire : if (count == 1 || count == 2)
            update(cell_content, electhead);
    }
}
Conclusion

- The cellular automata model is an efficient way to achieve scalable performance on parallel computers.
- Parallel cellular processing provides
  - A viable mathematical way to represent problems
  - The scalable performance of parallel processors
- Parallel cellular automata tools provide assistance in formulating problems, running models, and analyzing results.
Questions?