Parallelization and performance tests of the large scale LBM-CA solidification model

Bohumir Jelinek
Postdoctoral Fellow
CAVS, Mississippi State University

Columbus, 11/27/2012
Why LBM-CA?

When fluid flow around solidifying dendrites is considered, lattice Boltzmann method is faster than alternatives.
Lattice-Boltzmann method (LBM) calculates values of a quantity of interest at regularly spaced nodes governed by a partial differential equation subject to given boundary conditions.

Acknowledgment: Mohsen Eshraghi provided excellent code guide and parallelization suggestions.
LBM:

\[ f_i(r + e_i \Delta t, t + \Delta t) = f_i(r, t) + \frac{1}{\tau} \left( f_{eq}^{i}(r, t) - f_i(r, t) \right) \]

- Collision and streaming step
- Distribution functions \( f_i \) in each of the lattice direction \( e_i \) representing portion of the particles moving in that direction
### LBM parallelization

<table>
<thead>
<tr>
<th>CPU 1</th>
<th>CPU 2</th>
<th>CPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 4</td>
<td>CPU 5</td>
<td>CPU 6</td>
</tr>
<tr>
<td>CPU 7</td>
<td>CPU 8</td>
<td>CPU 9</td>
</tr>
</tbody>
</table>

Spatial domain decomposition
LBM parallelization – streaming

Direction
• horizontal (W, E)
• vertical (N, S)
• diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

Direction
• horizontal (W, E)
• vertical (N, S)
• diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Direction
• horizontal (W, E)
• vertical (N, S)
• diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<table>
<thead>
<tr>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal (W, E)</td>
</tr>
<tr>
<td>vertical (N, S)</td>
</tr>
<tr>
<td>diagonal (NW, NE, SW, SE)</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
LBM parallelization – streaming

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
### LBM parallelization – streaming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Direction**
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

Direction
• horizontal (W, E)
• vertical (N, S)
• diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Direction**
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
**LBM parallelization – streaming**

- **Direction**
  - horizontal (W, E)
  - vertical (N, S)
  - diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
## LBM parallelization – streaming

### Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<table>
<thead>
<tr>
<th>Direction</th>
<th>horizontal (W, E)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>vertical (N, S)</td>
</tr>
<tr>
<td></td>
<td>diagonal (NW, NE, SW, SE)</td>
</tr>
</tbody>
</table>
LBM parallelization – streaming

**Direction**
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
LBM parallelization – streaming

<table>
<thead>
<tr>
<th></th>
<th>CPU 7</th>
<th>CPU 8</th>
<th>CPU 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 4</td>
<td></td>
<td>CPU 5</td>
<td>CPU 6</td>
</tr>
<tr>
<td>CPU 1</td>
<td>CPU 2</td>
<td>CPU 3</td>
<td></td>
</tr>
</tbody>
</table>

Direction
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)
**LBM parallelization – streaming**

<table>
<thead>
<tr>
<th>CPU 7</th>
<th>CPU 8</th>
<th>CPU 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU 4</th>
<th>CPU 5</th>
<th>CPU 6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU 1</th>
<th>CPU 2</th>
<th>CPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Direction**
- horizontal (W, E)
- vertical (N, S)
- diagonal (NW, NE, SW, SE)

```python
buffer=send
MPI_Sendrecv_replace(
    buffer, dst=6, src=4)
recv=buffer
```
LBM parallelization – streaming

- Horizontal (W, E)
- Vertical (N, S)
- Diagonal (NW, NE, SW, SE)

```c
buffer = send
MPI_Sendrcv_replace(
    buffer, dst=2, src=8)
recv = buffer
```
LBM parallelization – streaming

Direction
• horizontal (W, E)
• vertical (N, S)
• diagonal (NW, NE, SW, SE)

buffer=send
MPI_Sendrcv_replace(buffer, dst=3, src=7)
recv=buffer
LBM parallelization – streaming

Street flow – example LBM problem:
velocity of flow
LBM parallelization

Dendrite growth in AlCu alloy upon cooling: temperature, velocity of flow, and solute concentration
LBM-CA solidification model – $C_l$, $v$, $T$

Flow of solute between solidifying dendrites in variable temperature field. Cooled at front and back boundaries, heated from left (inlet) and right (outlet) boundaries.
For dendrite growth, information from neighboring nodes is needed to update local node value.
LBM parallelization – ghost nodes

<table>
<thead>
<tr>
<th>CPU 7</th>
<th>CPU 8</th>
<th>CPU 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 4</td>
<td>CPU 5</td>
<td>CPU 6</td>
</tr>
<tr>
<td>CPU 1</td>
<td>CPU 2</td>
<td>CPU 3</td>
</tr>
</tbody>
</table>

Populate **ghost nodes** after each local update
LBM parallelization – ghost nodes

<table>
<thead>
<tr>
<th>CPU 7</th>
<th>CPU 8</th>
<th>CPU 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU 4</td>
<td>CPU 5</td>
<td>CPU 6</td>
</tr>
<tr>
<td>CPU 1</td>
<td>CPU 2</td>
<td>CPU 3</td>
</tr>
</tbody>
</table>

Populate ghost nodes after each local update east

MPI_Sendrcv( send, recv, dst=6, src=4)
LBM parallelization – ghost nodes

Populate ghost nodes after each local update west

MPI_Sendrecv(
  send, recv,
  dst=4, src=6)
LBM parallelization – ghost nodes

Populate ghost nodes after each local update north

MPI_Sendrcv(send, recv, dst=8, src=2)
LBM parallelization – ghost nodes

Populate ghost nodes after each local update south

MPI_Sendrcv(
  send, recv,
  dst=2, src=8)

Populate ghost nodes after each local update south
LBM parallelization – ghost nodes

Populate ghost nodes after each local update south-east

MPI_Sendrcv(send, recv, dst=3, src=7)
LBM parallelization – ghost nodes

Populate ghost nodes after each local update north-west

MPI_Sendrecv(
  send, recv,
  dst=7, src=3)
LBM parallelization – ghost nodes

Populate ghost nodes after each local update south-west

MPI_Sendrcv(send, recv, dst=1, src=9)
LBM parallelization – ghost nodes

- Populate ghost nodes after each local update north-east

MPI_Sendrecv(
  send, recv,
  dst=9, src=1)
LBM parallelization

3D Dendrite growth in AlCu alloy upon cooling: temperature, fluid flow, and solute concentration
LBM parallelization

3D Dendrite growth in AlCu alloy upon cooling: temperature, fluid flow, and solute concentration

simulation by Mohsen Eshraghi
Generating an initial configuration for parallel scaling tests

Simulation domain:
- rectangular lattice, 8000x6000 grid points
- dimensions: 2.4 mm x 1.8 mm (0.3 µm/lattice distance)
- 3264 random dendrite nucleation sites
- constant cooling rate 100K/s across the whole domain
- forced melt flow through inlet (left) and outlet (right) boundaries
- almost 16 GB of memory = single node of Kraken
- 400k time steps
- took about 10 hours on 192 cores on Talon @ MSU
Magnified portion of initial configuration
Growing dendrites to initial configuration
Generating an initial configuration for parallel scaling tests

Simulation domain:
- rectangular lattice, 8000x6000 grid points
- dimensions: 2.4 mm x 1.8 mm (0.3 μm/lattice distance)
- 3264 random dendrite nucleation sites
- constant cooling rate 100K/s across the whole domain
- forced melt flow through inlet (left) and outlet (right) boundaries
- almost 16 GB of memory = single node of Kraken
- 400k time steps
- took about 10 hours on 192 cores on Talon @ MSU
Speed up

- Speed up (strong scaling) represents how much faster a task is solved utilizing multiple cores.

- Speed up tests were performed by restarting simulation from the step when the dendrites were fairly grown in the incubation domain.

- Incubation domain is “split” equally between varying number of cores, then executed for 587 time steps with a flow forced at the inlet (left) and outlet (right), and with a specified cooling flow rate at all boundaries.
Speed up - constant task, 1 core
Speed up - constant task, 2 cores
Speed up - constant task, 4 cores
Speed up - constant task, 12 cores

<table>
<thead>
<tr>
<th>Core 1</th>
<th>Core 2</th>
<th>Core 3</th>
<th>Core 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 5</td>
<td>Core 6</td>
<td>Core 7</td>
<td>Core 8</td>
</tr>
<tr>
<td>Core 9</td>
<td>Core 10</td>
<td>Core 11</td>
<td>Core 12</td>
</tr>
</tbody>
</table>
Speed up - constant task, 12 cores
Speed up - constant task, 24 cores
Speed up - results

- strong scaling (speed up) near perfect up to 3072 cores
- Algorithm is memory bandwidth limited on multi-core architecture (low FLOP/byte ratio)
Scale up

- Scale up (weak scaling) tests checks if the algorithm can solve larger tasks when more cores are utilized without a significant performance penalty.

- Scale up tests were initialized from the stage when the dendrites were fairly grown in the incubation domain.

- Incubated domain was “duplicated” equally onto varying number of nodes, then executed for 587 time steps with a flow forced at the inlet (left) and outlet (right), and with a specified cooling flow rate at all boundaries.
Scale up - constant domain per node
Scale up - constant domain per node

Duplication of the incubation domain onto 4 nodes
Scale up - constant domain per node

Duplication of the incubation domain onto 4 nodes
## Scale up - constant domain per node

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 5</td>
<td>Node 6</td>
<td>Node 7</td>
<td>Node 8</td>
</tr>
<tr>
<td>Node 9</td>
<td>Node 10</td>
<td>Node 11</td>
<td>Node 12</td>
</tr>
<tr>
<td>Node 13</td>
<td>Node 14</td>
<td>Node 15</td>
<td>Node 16</td>
</tr>
</tbody>
</table>

Duplication of the incubation domain onto 16 nodes.
Scale up - results

Goal: constant calculation time

Graph showing calculation time in seconds for different numbers of cores, comparing Kraken and Talon.
Scale up - results

Demonstrated nearly perfect scale up

Largest domain:
- 41472 cores of Kraken
- over 165 billion grid nodes
- 11 millions of dendrites (only hundreds reported before)
- solute diffusion, melt convection, and heat transport
- dimensions 17.28 cm x 8.64 cm
- 587 time steps
- 40 minutes of simulation time
Computational resources

Talon, MSU HPC²:
- 3072 cores, 12 cores/node (user limit 192 cores / job)
- Intel Xeon X5660 @2.8GHz (Westmere) processors
- 24 GByte/node memory
- Voltaire quad data-rate InfiniBand (40Gb/s)
- peak performance of over 34.4 TeraFLOPS

Kraken, NICS/ORNL:
- 112,896 cores, 12 cores/node (user limit cores / job)
- AMD Opteron (Istanbul) @2.6GHz (Istanbul) processors
- 16 GByte/node memory
- Cray SeaStar2+ router
- peak performance of 1.17 PetaFLOPS
### XSEDE allocations

1) Simulations for 2D and 3D dendrite growth during alloy solidification:
   - SDSC-GORDON 250 kSU
   - NICS-KRAKEN 249 kSU
   - TACC-LONESTAR 1 kSU
   - ECSS

2) Large scale 3D modeling of microstructural evolution during alloy solidification
   - SDSC-GORDON 500 kSU
   - NICS-NAUTILUS 10 kSU
Progress on the parallelization of the 3D LBM/CA code

Already implemented
- Solute concentration and dendrite growth in parallel

Recently added
- Convection (fluid velocity) in parallel
- Output stride to reduce data for visualization
- Velocity calculation on a coarser subgrid
- Measure tip velocity and solute concentration profiles

Planned (as needed)
- Parallelize full temperature field calculation in 3D
Growth of Al-Cu dendrites in a 120x120x120 µm³ with 4.5 °C undercooling. From left to right, after 3, 7, 10, and 15 ms [1].

Columnar dendrites growing in an undercooled melt of Al-3wt%Cu. Domain size 180x180x144 (µm)³

By Mohsen Eshraghi
Published:

Journal articles:

In progress:

Journal articles:
- Eshraghi, M., Felicelli, S. D., Jelinek, B. A three-dimensional lattice Boltzmann-cellular automaton model for dendritic solidification under convection
Publications

Presentations, accompanied by articles in proceedings:

Planned:
Journal articles:
- Jelinek, B., Eshraghi, M., Felicelli, S. D., Peters, J. F. Large scale parallel lattice Boltzmann - cellular automaton model of three-dimensional dendritic growth
Conclusions

Accomplishments

- Implemented tests of the strong and weak parallel scaling of LBM/CA model with dendrites at advanced growth stage
- Parallelized velocity in the 3D lattice Boltzmann / cellular automaton model for dendrite growth
- 3D velocity calculation on a coarser subgrid
- Measuring tip velocity and solute concentration profiles

Plans

- Implement LBM-DEM coupling (in progress)