Writing parallel programs
Writing parallel programs

- Decide the serial algorithm first
Writing parallel programs

- Decide the serial algorithm first

SPMD model
Writing parallel programs

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement

SPMD model
Writing parallel programs

• Decide the serial algorithm first

• Data: how to distribute data among threads/processes?
  • Data locality: assignment of data to specific processes to minimize data movement

• Computation: how to divide work among threads/processes?
Writing parallel programs

- Decide the serial algorithm first
- Data: how to distribute data among threads/processes?
  - Data locality: assignment of data to specific processes to minimize data movement
- Computation: how to divide work among threads/processes?
- Figure out how often communication will be needed

SPMD model
Conway’s Game of Life

- Two-dimensional grid of (square) cells
- Each cell can be in one of two states: live or dead
- Every cell only interacts with its eight nearest neighbors
- In every generation (or iteration or time step), there are some rules that decide if a cell will continue to live or die or be born (dead $\rightarrow$ live)

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

By Lev Kalmykov - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=43448735
Conway’s Game of Life

- Two-dimensional grid of (square) cells
- Each cell can be in one of two states: live or dead
- Every cell only interacts with its eight nearest neighbors
- In every generation (or iteration or time step), there are some rules that decide if a cell will continue to live or die or be born (dead $\rightarrow$ live)

https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

By Lev Kalmykov - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=43448735
Two-dimensional stencil computation

- Commonly found kernel in computational codes
- Heat diffusion, Jacobi method, Gauss-Seidel method

Two-dimensional stencil computation

- Commonly found kernel in computational codes
- Heat diffusion, Jacobi method, Gauss-Seidel method

\[
\]
for(int t=0; t<num_steps; t++) {
    ...

    for(i ...) 
        for(j ...) 

    // copy contents of A_new into A
    ...
}
for(int t=0; t<num_steps; t++) {
    ...
    for(i ...) {
        for(j ...) {
        }
    }
    // copy contents of A_new into A
    ...
}
2D stencil computation in parallel
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes

![Diagram showing 2D grid with rows divided among processes]
2D stencil computation in parallel

- 1D decomposition
  - Divide rows (or columns) among processes

Ghost cells
2D stencil computation in parallel

• 1D decomposition
  • Divide rows (or columns) among processes

• 2D decomposition
  • Divide both rows and columns (2d blocks) among processes
2D stencil computation in parallel

- **1D decomposition**
  - Divide rows (or columns) among processes

- **2D decomposition**
  - Divide both rows and columns (2d blocks) among processes

Ghost cells
Prefix sum

- Calculate sums of prefixes (running totals) of elements (numbers) in an array
- Also called a “scan” sometimes

\[ p\text{Sum}[0] = A[0] \]

\[
\text{for}(i=1; \ i<N; \ i++) \ 
\quad \text{p\text{Sum}}[i] = \text{p\text{Sum}}[i-1] + A[i]
\]

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>pSum</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>15</td>
<td>21</td>
<td>...</td>
</tr>
</tbody>
</table>
Parallel prefix sum

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Parallel prefix sum

<table>
<thead>
<tr>
<th>Processes/threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
# Parallel prefix sum

<table>
<thead>
<tr>
<th>Processes/threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

**Stride 1**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>10</td>
<td>11</td>
<td>8</td>
<td>12</td>
<td>11</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>
Parallel prefix sum

<table>
<thead>
<tr>
<th>Processes/threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

Stride 1

|                  | 2 | 10 | 11 | 8 | 12 | 11 | 5 | 7 |

Stride 2

|                  | 2 | 10 | 13 | 18 | 23 | 19 | 17 | 18 |
## Parallel prefix sum

<table>
<thead>
<tr>
<th>Processes/threads</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

**Stride 1**

|                  | 2 | 10 | 11 | 8 | 12 | 11 | 5 | 7 |

**Stride 2**

|                  | 2 | 10 | 13 | 18 | 23 | 19 | 17 | 18 |

**Stride 4**

|                  | 2 | 10 | 13 | 18 | 25 | 29 | 30 | 36 |
In practice
In practice

- You have $N$ numbers and $p$ processes, $N >> p$
In practice

• You have $N$ numbers and $p$ processes, $N >> p$

• Assign a $N/p$ block to each process
  
  • Do the serial prefix sum calculation for the blocks owned on each process locally
In practice

- You have $N$ numbers and $p$ processes, $N \gg p$

- Assign a $N/p$ block to each process
  - Do the serial prefix sum calculation for the blocks owned on each process locally

- Then do parallel algorithm with partial prefix sums (using the last element from each local block)
  - Last element from sending process is added to all elements in receiving process’ sub-block
The \( n \)-body problem

- Simulate the motion of celestial objects interacting with one another due to gravitational forces
- Naive algorithm: \( O(n^2) \)
  - Every body calculates forces pair-wise with every other body (particle)

Data distribution in $n$-body problems

- Naive approach: Assign $n/p$ particles to each process
- Other approaches?
Data distribution in $n$-body problems

- Naive approach: Assign $n/p$ particles to each process
- Other approaches?

http://datagenetics.com/blog/march2013/
https://en.wikipedia.org/wiki/Z-order_curve
Data distribution in $n$-body problems

- Naive approach: Assign $n/p$ particles to each process
- Other approaches?

Space-filling curves

http://datagenetics.com/blog/march222013/
https://en.wikipedia.org/wiki/Z-order_curve
Data distribution in $n$-body problems

• Naive approach: Assign $n/p$ particles to each process

• Other approaches?

Space-filling curves

http://datagenetics.com/blog/march22013/
https://en.wikipedia.org/wiki/Z-order_curve

Data distribution in $n$-body problems

- Let us consider a two-dimensional space with bodies/particles in it
Data distribution in *n*-body problems

- Let us consider a two-dimensional space with bodies/particles in it
Data distribution in $n$-body problems

- Let us consider a two-dimensional space with bodies/particles in it.
Load balance and grain size

• Load balance: try to balance the amount of work (computation) assigned to different threads/processes
  • Bring ratio of maximum to average load as close to 1.0 as possible
  • Secondary consideration: also load balance amount of communication

• Grain size: ratio of computation-to-communication
  • Coarse-grained (more computation) vs. fine-grained (more communication)