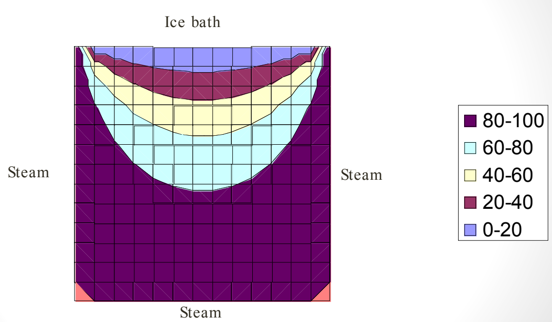
Domain Decomposition

Thomas Hauser
Thomas.hauser@colorado.edu

Steady State Heat Distribution Problem



Solving the Problem

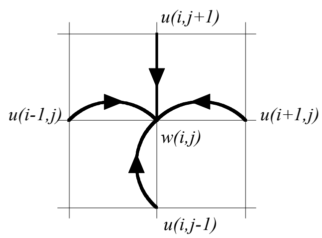
- Underlying PDE is the Poisson equation

$$u_{xx} + u_{yy} = f(x, y)$$

- This is an example of an elliptical PDE
- Will create a 2-D grid
- Each grid point represents value of state state solution at particular (*x*, *y*) location in plate

Heart of Sequential C Program

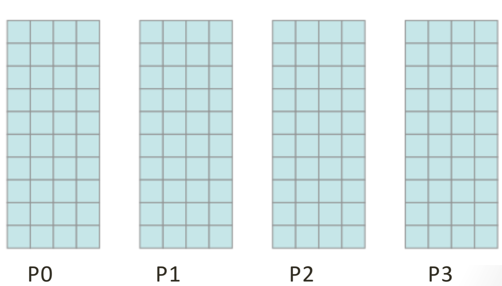
```
w[i][j] = (u[i-1][j] + u[i+1][j] +
           u[i][j-1] + u[i][j+1]) / 4.0;
```



Parallel Algorithm 1

- Associate primitive task with each matrix element
- Agglomerate tasks in contiguous rows (rowwise block striped decomposition)
- Add rows of ghost points above and below rectangular region controlled by process

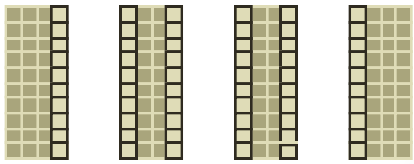
Agglomerate and map



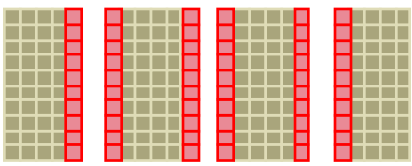
How does the communication work?

Communication Still Needed

- Exchange between columns
- Values in black cells cannot be computed without access to values held by other tasks



Matrices Augmented with Ghost Points

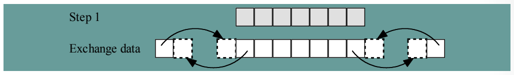


Red cells are the ghost points.

Ghost Points

- Ghost points: memory locations used to store redundant copies of data held by neighboring processes
- Allocating ghost points as extra columns simplifies parallel algorithm by allowing same loop to update all cells

Communication to put ghost points in place

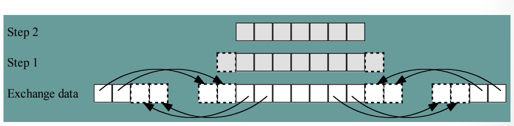


Ghost Point MPI Communication

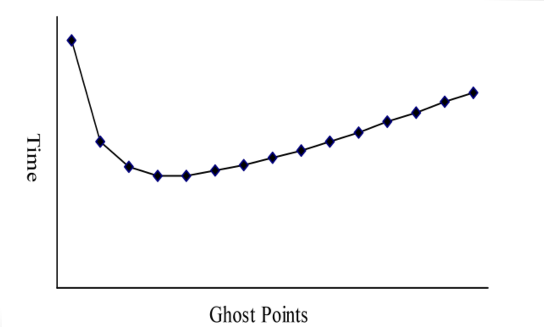
- Take 3 minutes to outline the MPI communication for exchanging ghost points between processors.
  - What MPI functions are necessary
  - Is there a way to overlap communication and computation?
  - Can you reduce communication by increasing computation?

Improve communication efficiency

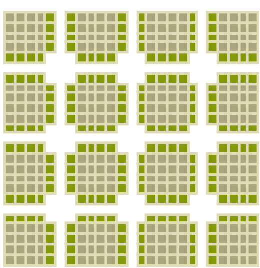
- Add more ghost points (second ghost column)
- Replicate data to reduce number of messages per computation



Communication time vs. number of ghost columns



Example Decomposition



Implementation Details

- Using ghost points around 2-D blocks requires extra copying steps
- Ghost points for left and right sides are not in contiguous memory locations
- An auxiliary buffer must be used when receiving these ghost point values
- Similarly, buffer must be used when sending column of values to a neighboring process

Data Decomposition Options

- Interleaved (cyclic)
  - Easy to determine "owner" of each index
- Block
  - Balances loads
  - More complicated to determine owner if *n* not a multiple of *p*

Block Decomposition Options

- Want to balance workload when *n* not a multiple of *p*
- Each process gets either $\lceil n/p \rceil$ or $\lfloor n/p \rfloor$ elements
- Seek simple expressions
  - Find low, high indices given an owner
  - Find owner given an index
- floor(*x*) = $\lfloor x \rfloor$ is the largest integer not greater than *x*
- ceiling(*x*) = $\lceil x \rceil$ is the smallest integer not less than *x*

Method #2

- Scatters larger blocks among processes
- First element controlled by process *i*

$$\lfloor in/p \rfloor$$

- Last element controlled by process *i*

$$\lfloor (i+1)n/p \rfloor - 1$$

- Process controlling element *j*

$$\lfloor p(j+1)-1)/n \rfloor$$

Examples

17 elements divided among 7 processes

17 elements divided among 5 processes

17 elements divided among 3 processes

Illustration

- Illustrate how block decomposition method #2 would divide 13 elements among 5 processes.
  - First element controlled by process *i*

$$\lfloor in/p \rfloor$$

13(0)/ 5 = 0   13(2)/ 5 = 5   13(4)/ 5 = 10

13(1)/5 = 2   13(3)/ 5 = 7

Block Decomposition Macros

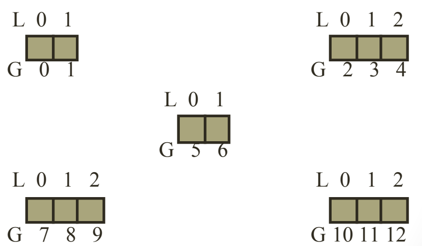
```
#define BLOCK_LOW(id,p,n)   ((i)*(n)/(p))

#define BLOCK_HIGH(id,p,n) \
        (BLOCK_LOW((id)+1,p,n)-1)

#define BLOCK_SIZE(id,p,n) \
        (BLOCK_LOW((id)+1)-BLOCK_LOW(id))

#define BLOCK_OWNER(index,p,n) \
        (((p)*(index)+1)-1)/(n))
```

Local vs. Global Indices



Looping over Elements

- Sequential program

```
for (i = 0; i < n; i++) {
    …
}
```

- Parallel program

Index *i* on this process…

```
size = BLOCK_SIZE (id,p,n);
for (i = 0; i < size; i++) {
  gi = i + BLOCK_LOW(id,p,n);
}
```

…takes place of sequential program's index *i*