Center for Computation & Technology

Information Technology Services

Parallel Programming Workshop

Brought to you by

Le Yan, Wei Feinstein,
Feng Chen, Xiaoxu Guan and Jim Lupo

Extreme Science and Engineering Discovery Environment

Registration

- **Please make sure you're signed in.**
- **Won't need a computer this morning**
  - ***unless you need a calculator to add integers***

Important Concepts

- Decomposition
- Scaling
- Speedup

We will jointly “discover” the meaning of these terms through experiment and group exercises – ease into programming only when necessary.

Distributed Memory Programming

- Two main models for doing parallel programming:
- Distributed Memory – workers must talk with one another to get data.
- Shared Memory – Workers view the same memory space.

Each has different issues.

Take on Distributed Memory first.

The Data Set

- Any confusion over the terms "integer" and "real" numbers?
- The data at hand consists of:
  - 50 data cards.
  - 5 integer numbers per card.
  - An integer card identifier.

Set: 14

164	**5**	**76**	**144**	**105**

Exercise 1

- Desired analysis: summation over 4 cards
- Divide into groups.
- Each group needs a time keeper.

Pay attention to the process!

Exercise 1 Outcomes

- What was the basic “unit of work” or task?
- What discreet steps were involved?

Yea verily, computers are lowly beasts and must be instructed tediously.

Exercise 1 Summary

- Process had 3 distinct steps:
  - Hand out cards
  - Sum the numbers
  - Report results
- More formally:
  - Distribute work (tasks).
  - Perform work
  - Gather results

Exercise 2 – Two Workers

- Repeat Ex 1, only with 2 people adding numbers.
- What changes?

Added Workers

- What happened with more workers?
- The process changes a little:
  - Distribute work
    - How to do that? **Communicate!**
  - Perform work
  - Gather results
    - Gather partial results. **Communicate!**
    - Compute final result
    - Report result

Exercise 3

- What happens with 3 workers?
- What happens with 4 workers?
- Could we use more than 4 workers?

Exercise 3 Outcomes

- More workers => More communication
- Balanced work assignments?
- Task starvation? (run out of cards)
- How do the input and output compare with Ex 1?

Everybody's talking at me, I don't hear a word their say'ng ...*

**** Fred Neil***

Comment on Scaling

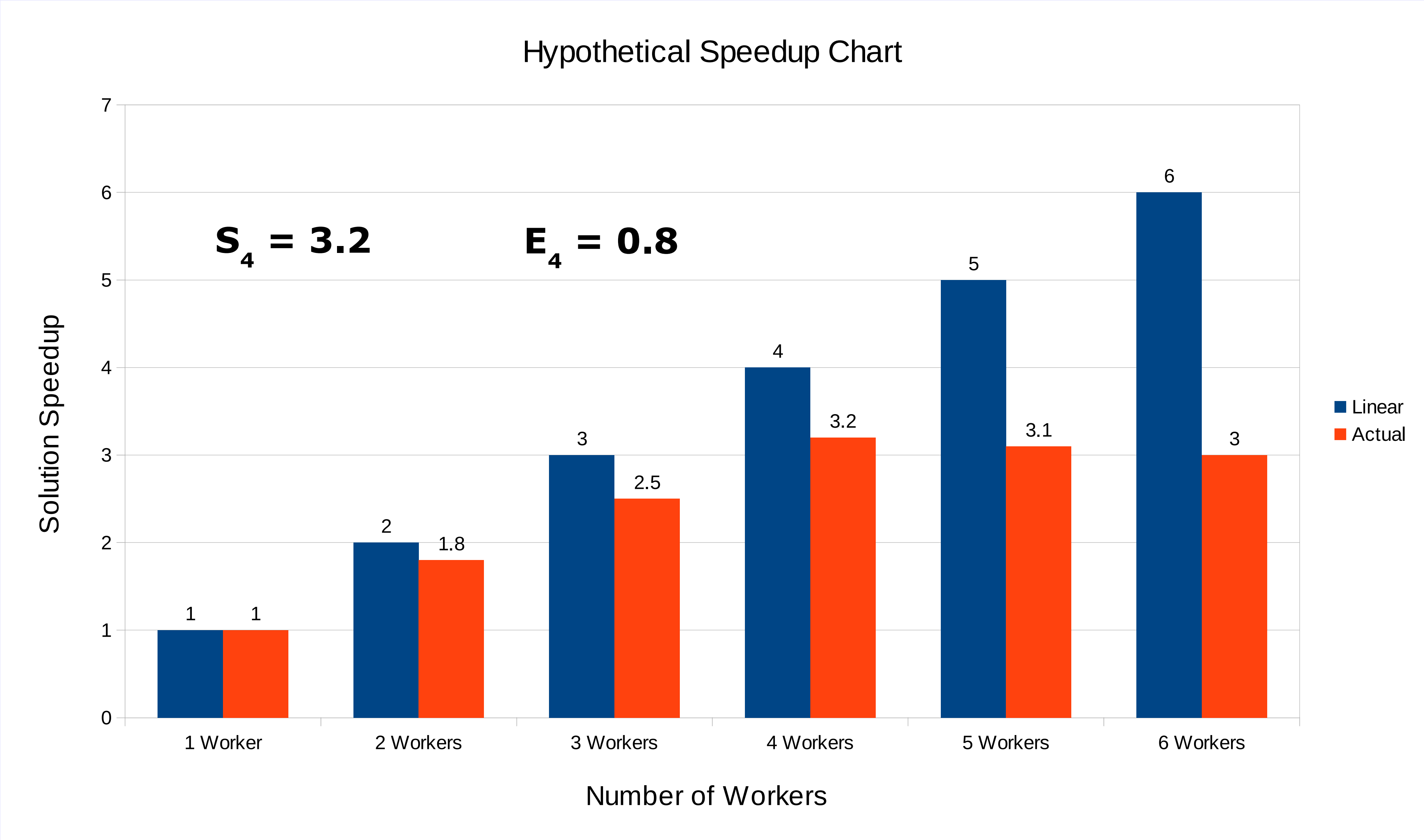
- How does parallel work speed up, i.e. "scale"?

$$S_p = \frac{T_1}{T_n} \qquad S_{serial} = \frac{T_{serial}}{T_n}$$

- How efficient is it? Again, two types:

$$E_p = \frac{T_1}{n T_n} \qquad E_{serial} = \frac{T_{serial}}{n T_n}$$

Beware of "Lies, damn lies, and statistics . . ."


Hypothetical Speedup Chart
S4 = 3.2
E4 = 0.8
Solution Speedup
0
1
2
3
4
5
6
7
1 Worker
2 Workers
3 Workers
4 Workers
5 Workers
6 Workers
Number of Workers
Linear
Actual
1
1
2
1.8
3
2.5
4
3.2
5
3.1
6
3

Overhead Expense

- 80% efficiency => 20% overhead.
  - If one hour on 5 computers, then 1 computer worth of power is unused!
- Constant trade-off between time-to-answer and expense, even if the usage seems "free".
- Time on most HPC systems is charged in core-hours (or *service units*), so low efficiency still *costs* more as service units are used up faster.

Distributing Work (Data)

- Shared data?
  - Each worker has a copy
  - Each worker has an ID
  - Use ID to *compute* what to work on.
- Distributed data?
  - Head worker has all the data.
  - Head worker knows # of workers.
  - Head worker computes decomposition.
  - Head worker *sends pieces* to workers.

Sharing Data

- Parallel file system – all workers see same data files.
- Broadcast – head worker broadcasts all data to all workers.

Considerations

- How much time is required to communicate?
- Does machines have access to shared file systems?

Concept Summary

When you approach problem to programming, ask yourself:

- What algorithm is required?
- How best to decompose the work?
- How is it suppose to scale?
- Minimize comm to get speedup.
- Test to see what has been achieved.

Shared Memory Programming

- Distributed Memory Programming recap:
  - Each worker was isolated.
  - Sent or computed work decomposition info.
  - Sent data or shared via file system.
- What changes with Shared Memory Programming?
  - Workers part of same system (i.e. cores).
  - Each worker can see all data in memory.
  - *Communication* replaced by *coordination* of read/write access.

Exercise 4

Assume all workers can see all the data - how does summation task change?

	A	B	C	D	E	Sums
1	6	3	13	78	35	
2	49	60	138	34	79	
3	59	108	108	188	110	
4	137	50	4	167	189	
5	83	136	215	26	140	
6	0	187	77	216	51	

Total

Exercise 4 Outcomes

- Benefits?
- Difficulties?

Concept Summary

- Shared memory lets all processors see all data, it is just there – no work to distribute it. BUT, need to coordinate changes!
- Shared Memory Model is growing in popularity as more cores per node become available, and new devices such as GPUs become common place – multi-core PCs use shared memory.
- Hybrid or Heterogeneous models are becoming important as the needed to combine Shared and Distributed models increase.

Parallel Thinking

- What kind of questions do you need to consider when approaching a new program?

- Algorithm – numerical stability? programmability?
- Data size – memory needs?
- Machine architecture – shared/distributed/both?
- Code lifetime – save FTE's or machine hours?
- Choice of language?
- Choice of tools?

Break

The Laplace Heat Equation

- For a "real" problem, consider how to go about solving the Laplace Heat Equation in 2-D. Idea is to determine the temperature at any point on a surface, given the temperature at the boundaries:

Center for Computation & Technology
Information Technology Services

Formal Solution

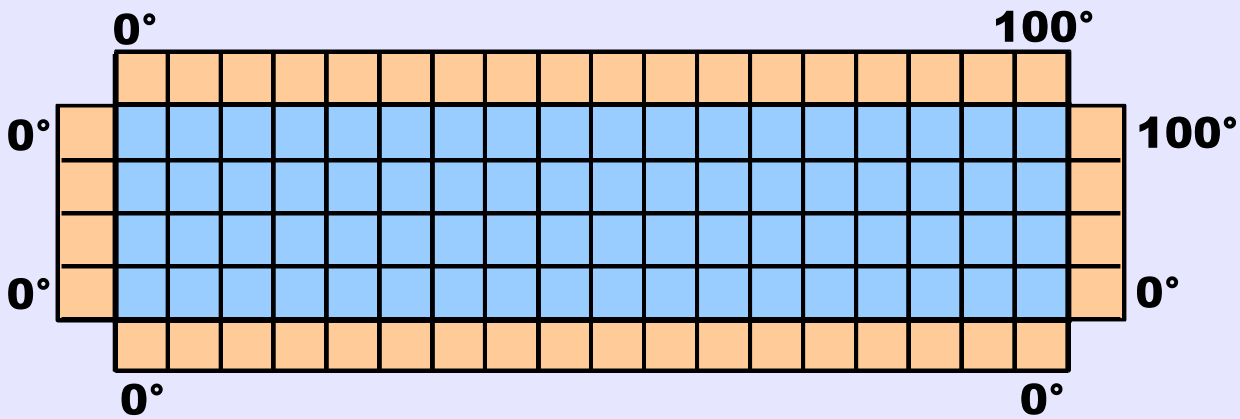
The solution must satisfy:

$$\nabla^2 \varphi = 0$$

with the application of Dirichlet boundary conditions (constant values around edge of region.

The Serial Solution

Subdivide the surface into a mesh of points, add boundary points.



Apply the following 5-*point stencil* iteratively until the temperature stops changing (new temp approximates old temp) to interior only:

$$T_{i,j}^{n+1} = 0.25 * \left(T_{i-1,j}^{n} + T_{i+1,j}^{n} + T_{i,j-1}^{n} + T_{i,j+1}^{n} \right)$$

Exercise 5: 1-D Problem

0	?	?	?	?	?	100

$$T_i^{n+1} = 0.5 * \left(T_{i-1}^n + T_{i+1}^n \right)$$

Discuss programming this problem in your group.

Step 1:

Array 1	0	0	0	0	0	0	100
Array 2	0	0	0	0	0	50	100

Step 2:

Array 1	0	0	0	0	25	50	100
Array 2	0	0	0	0	0	50	100

Step 3:

Array 1	0	0	0	0	25	50	100
Array 2	0	0	0	12.5	25	62.5	100

Exercise 5: Solution

70 iterations to reach 0.001% convergence bound.

0	16.6661	33.3324	49.9988	66.6658	83.3327	100

Exercise 6

0	?	?	?	?	?	?	100

Now the question is, how would we do this in parallel?

Need one small modification, so try using 2 workers first.

Process Start

Initialize:

0	0	0	0	0	0	0	100

Decompose:

0	0	0	0	0	Worker 1

0	0	0	0	100	Worker 2

"Ghost" or overlapped cells.

Process Iteration

Compute:

0	0	0	0	0	Worker 1

0	0	0	50	100	Worker 2

Communicate:

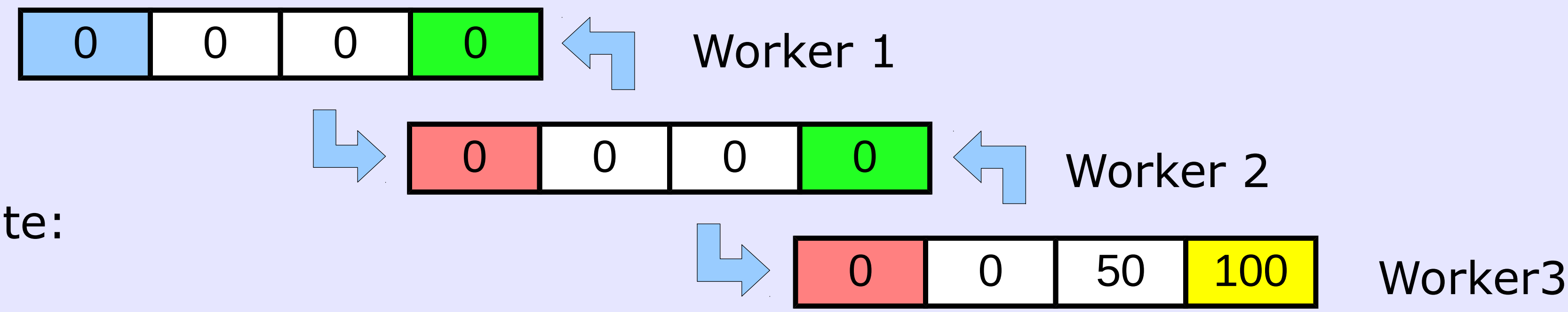
0	0	0	0	0	Worker 1

0	0	0	50	100	Worker 2

Lather, Rinse, Repeat.

What Would 3 Workers Involve?

Communicate:



Workers in the middle have to communicate intermediate results to neighbors on both sides!

Number of workers limited by problem size!

Serial Program

- Grab a copy of the program named:
  `/work/jalupo/laplace_solver_serial.f90`
- Open with “less” or “vi” so you can follow along.
- Anyone have trouble reading Fortran?
- Anyone not know how to compile and run a Fortran program?

Main Components

- **`program laplace_main`** – program main line.
- **`subroutine laplace`** – the actual solver. It also allocates memory to hold the 2-D mesh based on the requested rows and columns.
- **`subroutine initialize`** – sets the internal temperatures to 0.
- **`subroutine set_bcs`** – sets up the boundary conditions.

Compiling Fortran

- Here is a quick summary of how to compile and run this particular program (assumes default environment):

```
$ ifort -o laplace laplace_solver_serial.f90
$ ./laplace
```

- You should see the following line of text on your screen:

```
Usage: laplace nrows ncols niter iprint relerr
```

Now try executing the program with some real numbers:

```
$ ./laplace 100 200 3000 300 0.001
```

Results of Run

```
$ ./laplace 100 200 10000 3000 0.01

 Solution has converged.
Iterations:            2241
Max error:            0.01
Total time:           0.079s
```

What if the problem gets bigger, and error condition was changed to 0.001?

Higher Accuracy Run

```
$ ./laplace 1000 1000 30000 1000 0.001

 Solution has converged.
Iterations:           29812
Max error:            0.001
Total time:           60.546s
```

Why go to parallel?

What if this was only part of a simulation and the temperatures changed 25,000 times?

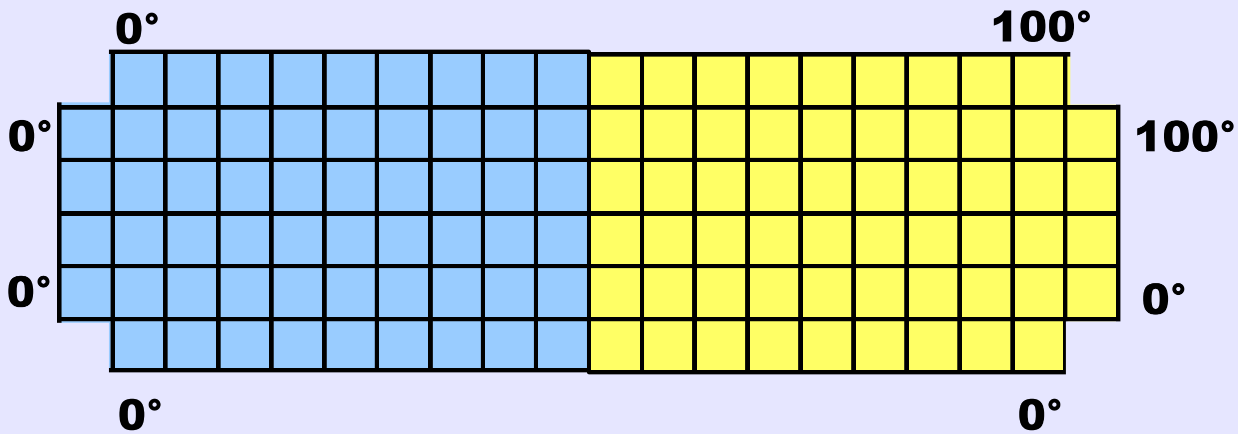
Even though 1 solution taking 1 second seems fast, 25,000 solutions would take 7 hours!

Can it be done in parallel to speed up the over all simulation time?

How do we approach the solution in parallel?

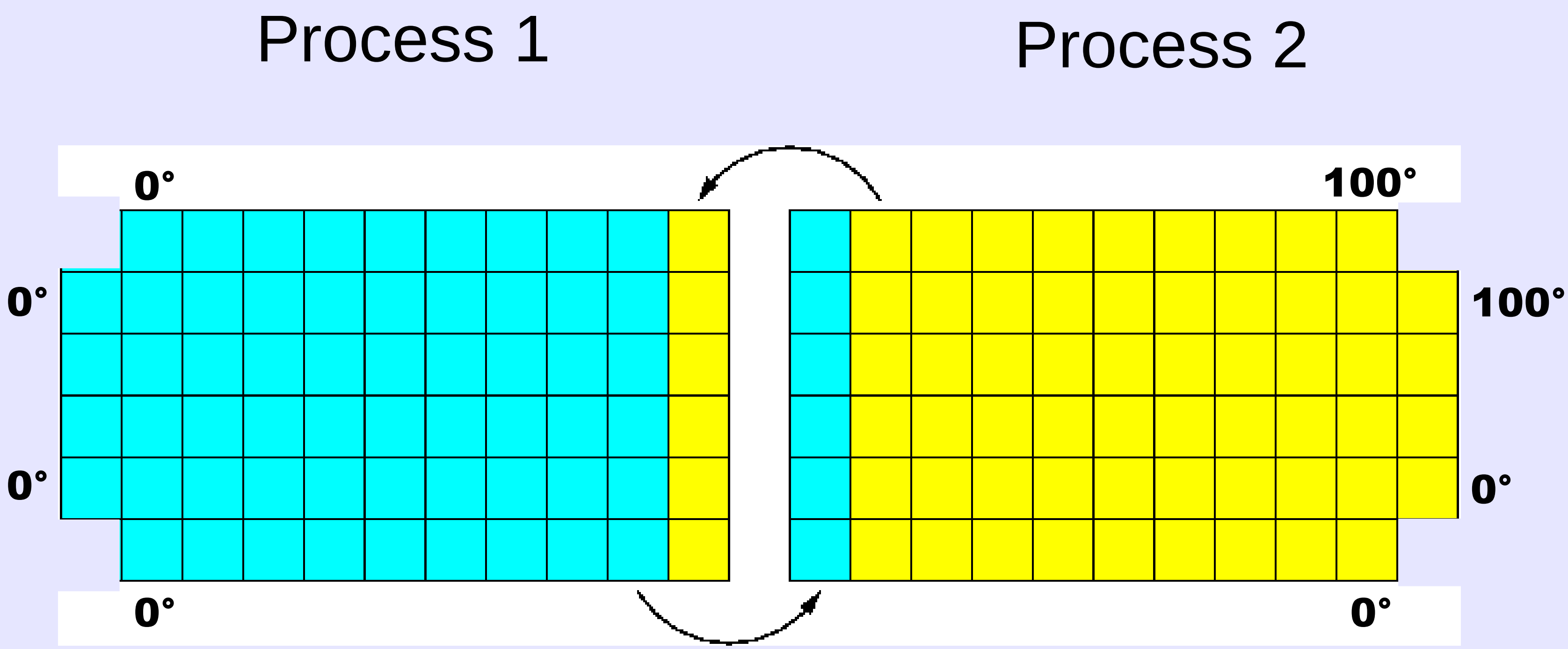
Decomposition

Assuming 2 processors, let's divide the surface in half.



What overhead do we have to consider adding to make this give the same answer?

Ghost Cells



Overhead

- Breaking up the problem so multiple processes can work on it introduces *overhead*:
  - Logic must be added so each process knows which part of the mesh it is expected to work on. This directly impacts how the code will start up.
  - Communication must be added so data from adjoining regions can be properly updated.
  - Code must be added so the final results can be communicated. This directly impacts how the code will report results and terminate.
- A serial program is not the same as a parallel program running on 1 processor!

Compute/Communication Bound

- Clearly, if you increase the number of processes working on this problem, the amount of communication required increases.
- With a few processes, this problem exhibits the property of being *compute bound*.
- When the number of processes approach the number of mesh points, it becomes *communication bound*.
- All parallel programs exhibit one form or the other depending on the problem specifics.

LUNCH