PARALLELISM IN CHAPEL, PART I

Chapel Team, edited by Michelle Strout

April 15, 2025

PLAN

- **Announcements**
  - SA7 grades are posted
  - LA3 is due on Friday April 25th (1.5 weeks left)
  - Final projects are due Friday May 2nd (2.5 weeks left)

- **Last time**
  - TopHat questions about Chapel basics
  - Data parallelism in Chapel
  - Domain decomposition in Chapel

- **Today**
  - TopHat questions about Chapel data parallelism
  - LA3 parallelism suggestions
  - Domain decomposition in Chapel

OUTLINE: OVERVIEW OF PARALLELISM IN CHAPEL, PART II

- LA3: Parallelizing a Chapel program
- Extra, Extra Credit: Leaderboard for performance improvement and/or UA HPC system
- Implicit Communication: Remote writes/Puts and Reads/Gets
- Parallelizing a 1D heat diffusion solver (Hands On)
- Heat 2D example with CommDiagnostics (Hands On)

LA3: PARALLELIZING A COMPUTATION IN CHAPEL

LA3: PARALLELIZING CHAPEL

Basics for LA3

- la3_parallel.chpl is a copy of la3_serial.chpl
- Improve the performance of la3_parallel.chpl so it is at least 10% faster than la3_serial.chpl
- Describe what you did in the comment header

Ideas

- Parallelize one or more loops
- Fuse the loops and remove temp file write and read
- Distributed data parallelism, probably need to run on the UofA HPC system
- Deal with load imbalance issues
- Limit number of tasks to when scaling improves

la3_serial.chpl

```
// convert all files to gray scale
for fname in files {
…
 var imageArray = readImage(fName,imageType.png);
 var grayImage = rgbToGrascale(imageArray);
 writeImage(grayImage,…, grayImage);
}


// do edge detection on all of the grayscale files
for fname in files {
…
 var grayArray = readImage(fName,imageType.png);
 var sobelImage = sobelEdgeDetection(grayArray);
 writeImage(edgefName,…, sobelImage);
}
```

Using the UofA HPC system for Extra, Extra Credit

- Log into HPC system and run a Chapel program

```
Laptop_prompt> ssh netid@hpc.arizona.edu
[netid@gatekeeper ~]$ shell
(puma) [netid@wentletrap ~]$ ocelote
(ocelote) [netid@wentletrap ~]$ interactive -a cs372spring2025 -n 8 -N 4 -t 1:0:0
Run "interactive -h for help customizing interactive use"
Submitting with /usr/local/bin/salloc --job-name=interactive --mem-per-cpu=4GB --
nodes=4    --ntasks=8 --time=1:0:0 --account=cs372spring2025 --partition=standard
// set up an ssh key for use with GitHub
// clone your LA3 github repository
// cd into a directory you have with Chapel code
(ocelote) [netid@i7n# Chapel]$ module load chapel-ibv
(ocelote) [netid@i7n# Chapel]$ chpl --version
chpl version 2.4.0
…
(ocelote) [netid@i7n# Chapel]$ chpl hello6-taskpar.chpl
(ocelote) [netid@i7n# Chapel]$ export GASNET_PHYSMEM_MAX=6GB
(ocelote) [netid@i7n# Chapel]$ ./hello6-taskpar -nl 2
```

- References

  - https://hpcdocs.hpc.arizona.edu/registration_and_access/system_access/#command-line-access

IMPLICIT COMMUNICATION:
REMOTE WRITES/PUTS AND READS/GETS

CHAPEL SUPPORTS A GLOBAL NAMESPACE WITH PUTS AND GETS
Note 1: Variables are allocated on the locale where the task is running

```
03-
onClause.chpl
config const verbose = false;
var total = 0,
    done = false;

…

on Locales[1] {
  var x, y, z: int;
  …



}
```

verbose	false
total	0
done	false

locale 0

x	0
y	0
z	0

locale 1

CHAPEL SUPPORTS A GLOBAL NAMESPACE WITH PUTS AND GETS
Note 2: Tasks can refer to lexically visible variables, whether local or remote

📄 03-onClause.chpl

03-onClause.chpl

```chapel
config const verbose = false;
var total = 0,
    done = false;

…

on Locales[1] {
  if !done {
    if verbose then
      writef("Adding locale 1's contribution");
    total += computeMyContribution();
  }
}
```

verbose false
total 0
done false

locale 0

code runs on locale 1, but refers to values stored on locale 0

```chapel
if !done {
  if verbose then
    writef("Adding…
  total += computi…
}
```
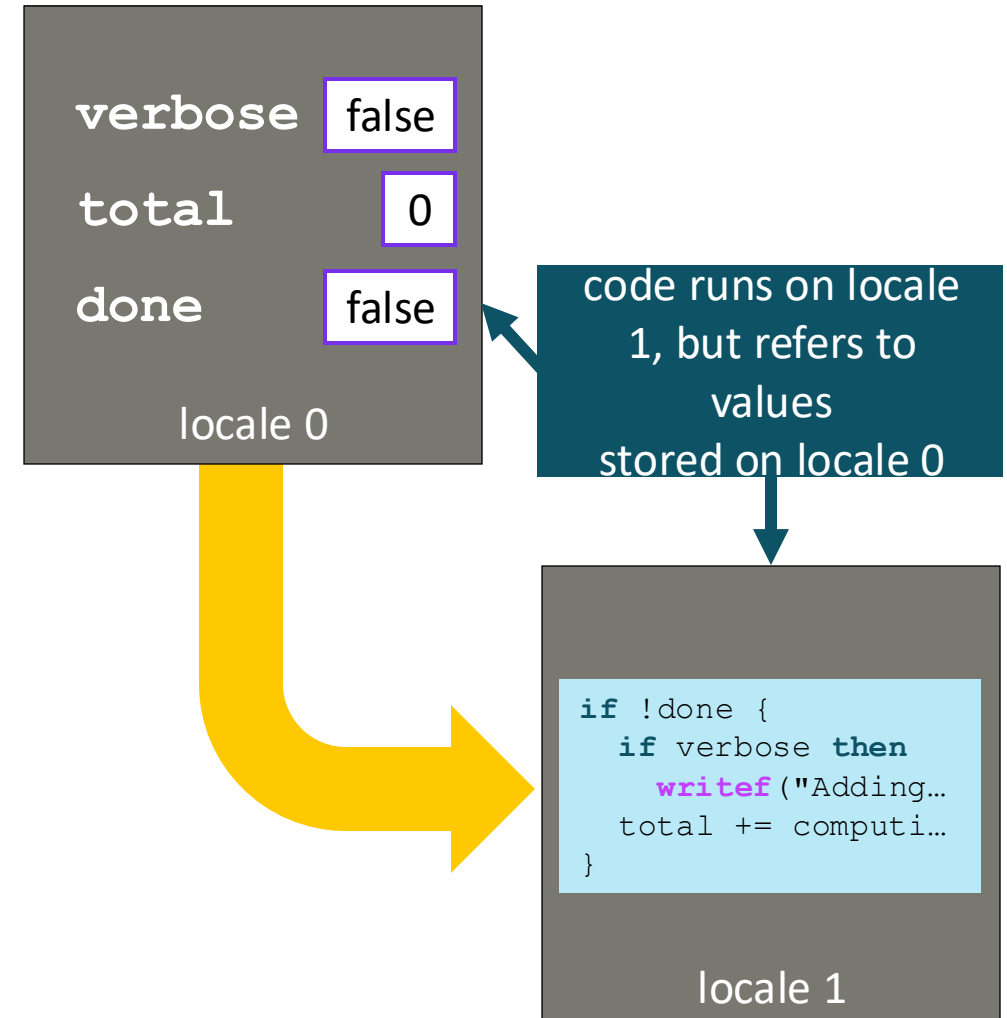
locale 1

ARRAY-BASED PARALLELISM AND LOCALITY

03-basics-distarr.chpl

```chapel
writeln("Hello from locale ", here.id);

var A: [1..2, 1..2] real;

use BlockDist;

var D = blockDist.createDomain({1..2, 1..2});
var B: [D] real;
B = A;
```

Chapel also supports distributed domains (index sets) and arrays

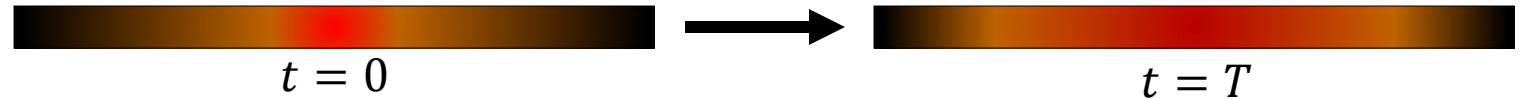**They also result in parallel distributed computation**
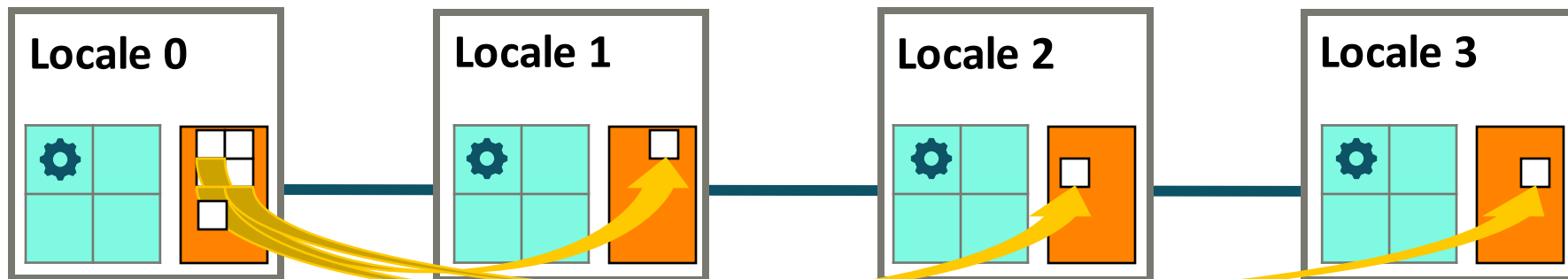
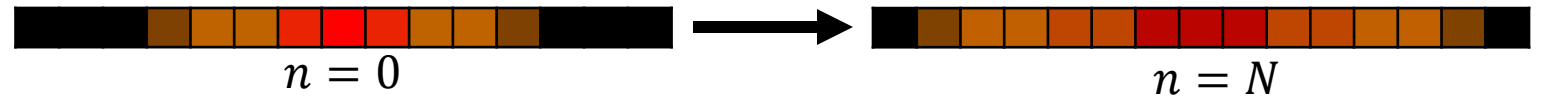# PARALLELIZING A 1D HEAT DIFFUSION SOLVER (HANDS ON)

Also read https://github.com/jeremiah-corrado/Chapel-Heat1D-PPA

1D HEAT EQUATION EXAMPLE

Differential equation: $\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$

$t = 0$ $t = T$

Discretized (finite difference) equation: $u_i^{n+1} = u_i^n + \alpha\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right)$

- where $i \in \Omega \subset \mathbb{R}^1$ are discrete points in space, and $(n, n+1, \dots)$ are discrete instances in time

$n = 0$ $n = N$

Finite difference algorithm:

- define Ω to be a set of discrete points along the x-axis
- define $\widehat{\Omega}$ over the same points, excluding the boundary
- define an array u to over Ω
- set some initial conditions
- create a temporary copy of u, named un
- for N timesteps:

  (1) swap u and un

  (2) compute u in terms of un over $\widehat{\Omega}$

```
1    const omega = {0..<nx},
2          omegaHat = omega.expand(-1);
3    var u: [omega] real = 1.0;
4    u[nx/4..3*nx/4] = 2.0;
5    var un = u;
6    for 1..N {
7      un <=> u;
8      forall i in omegaHat do
9        u[i] = un[i] + alpha *
10             (un[i-1] - 2*un[i] + un[i+1]);
11   }
```

1D HEAT EQUATION EXAMPLE

This pattern is often referred to as a *Stencil Computation*

- The values in the array can be computed by applying a "stencil" to its previous state
- Note that in this case, the stencil can be applied to the entire array in parallel
 - each value in un depends strictly on values in u

un

$n = 0$
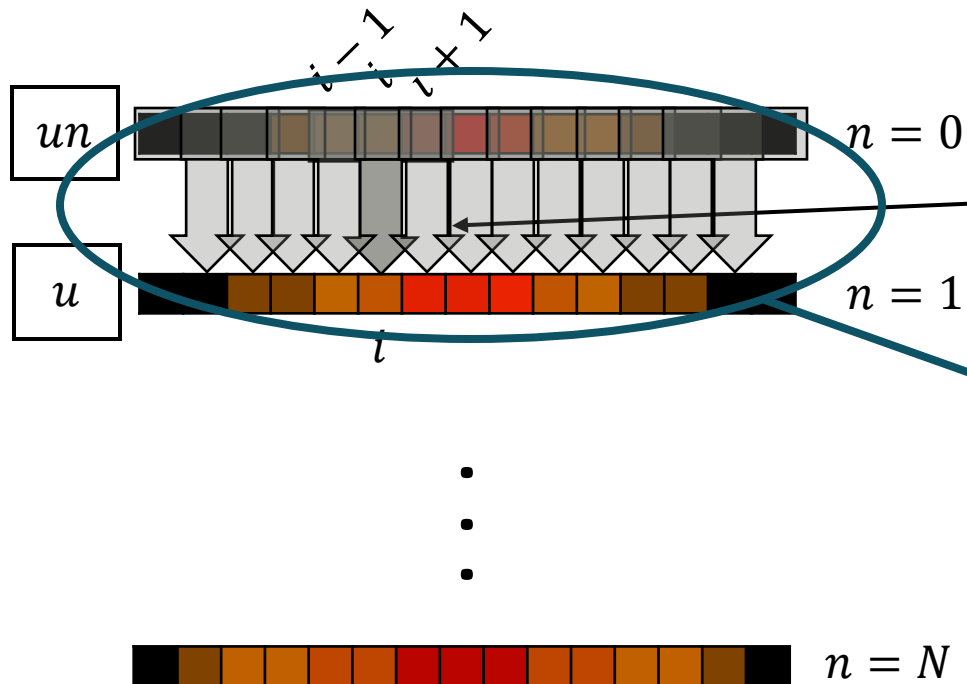
"stencil"

$$u_i^{n+1} = u_i^n + \alpha\,(u_{i-1}^n - 2u_i^n + u_{i+1}^n)$$

u

$n = 1$

i

$n = N$

```
 7   ...
 8      forall i in omegaHat do
 9         u[i] = un[i] + alpha *
10                   (un[i-1] - 2*un[i] + un[i+1]);
11   ...
```

HANDS ON: DISTRIBUTING THE 1D HEAT EQUATION

Imagine we want to simulate a very large domain
- We could use the Block distribution to distribute u and un across multiple locales
  - taking advantage of their memory and compute resources
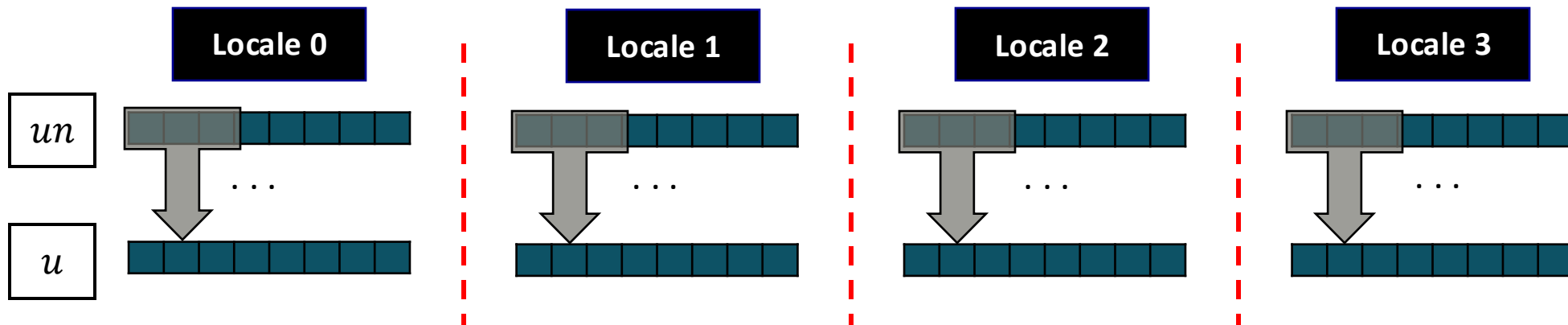
Look at **heat-1D-block.chpl** and fill in the blanks to make the arrays block-distributed

Hint | Define a block-distributed domain:

```
use BlockDist;
...
const myBlockDom = blockDist.createDomain({1..10});
```

HANDS ON: DISTRIBUTING THE 1D HEAT EQUATION

Solution: make 'omega' block-distributed:
```
omega = blockDist.createDomain({0..<nx});
```

Why does this work?
- 'omegaHat' inherits 'omega's distribution
- 'u' is block-distributed
- 'un' inherits 'u's domain (and distribution)
- 'omegaHat' invokes 'blockDist's parallel/distr. iterator
  - the body of the loop is automatically split across multiple tasks on each locale
- Communication occurs automatically when a loop references a value stored on a remote locale

```
1   const omega =
2         blockDist.createDomain({0..<nx}),
3         omegaHat = omega.expand(-1);
4   var u: [omega] real = 1.0;
5   u[nx/4..3*nx/4] = 2.0;
6   var un = u;
7   for 1..N {
8     un <=> u;
9     forall i in omegaHat do
10      u[i] = un[i] + alpha *
11            (un[i-1] - 2*un[i] + un[i+1]);
12  }
```

u[i+1]

un

u
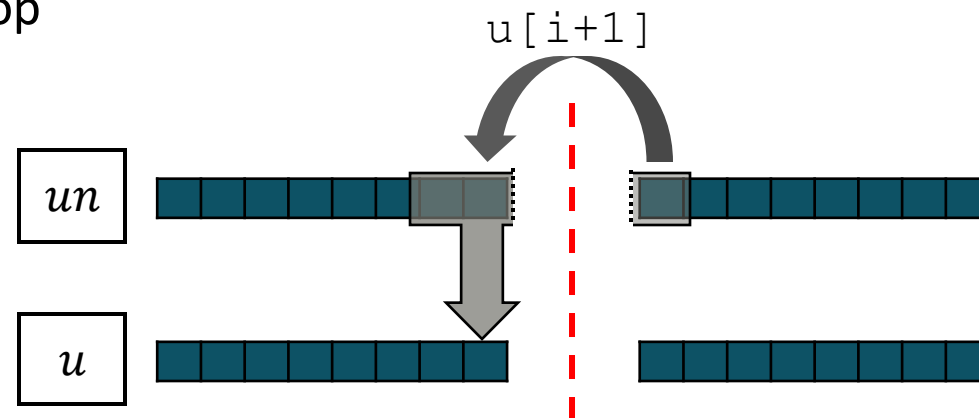
HEAT 2D EXAMPLE WITH COMMDIAGNOSTICS (HANDS ON)

2D HEAT EQUATION EXAMPLE

2D and 3D stencil codes are more common and practical
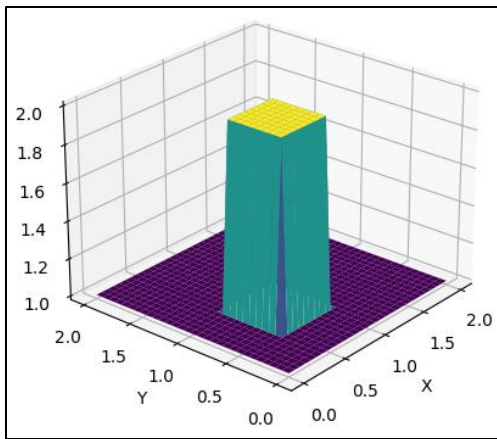 • They also present more interesting considerations for parallelization and distribution

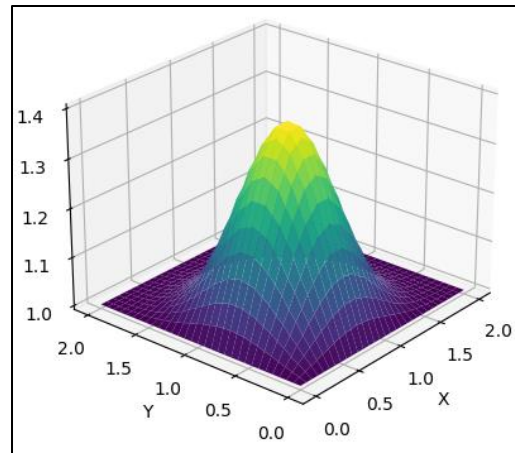**2D heat / diffusion PDE:**

$$\frac{\partial u}{\partial t} = \alpha \Delta u = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right)$$

Discretized (finite-difference) form:

$$u_{i,j}^{n+1} = u_{i,j}^n + \alpha \left(u_{i+1,j}^n + u_{i-1,j}^n - 4u_{i,j}^n + u_{i,j+1}^n + u_{i,j-1}^n \right)$$

$n = 0$

$n = N$

```
1   const omega = {0..<nx, 0..<ny},
2         omegaHat = omega.expand(-1);
3   var u: [omega] real = 1.0;
4   u[nx/4..3*nx/4] = 2.0;
5   var un = u;
6   for 1..N {
7     un <=> u
8     forall (i, j) in omegaHat do
9       u[i, j] = un[i, j] + alpha * (
10                 un[i-1, j] + un[i, j-1] +
11                 un[i+1, j] + un[i, j+1] -
12                 4 * un[i, j]);
13  }
```

PARALLEL 2D HEAT EQUATION

$$u^n$$

Stored in un

$$u^{n+1}$$

Stored in u
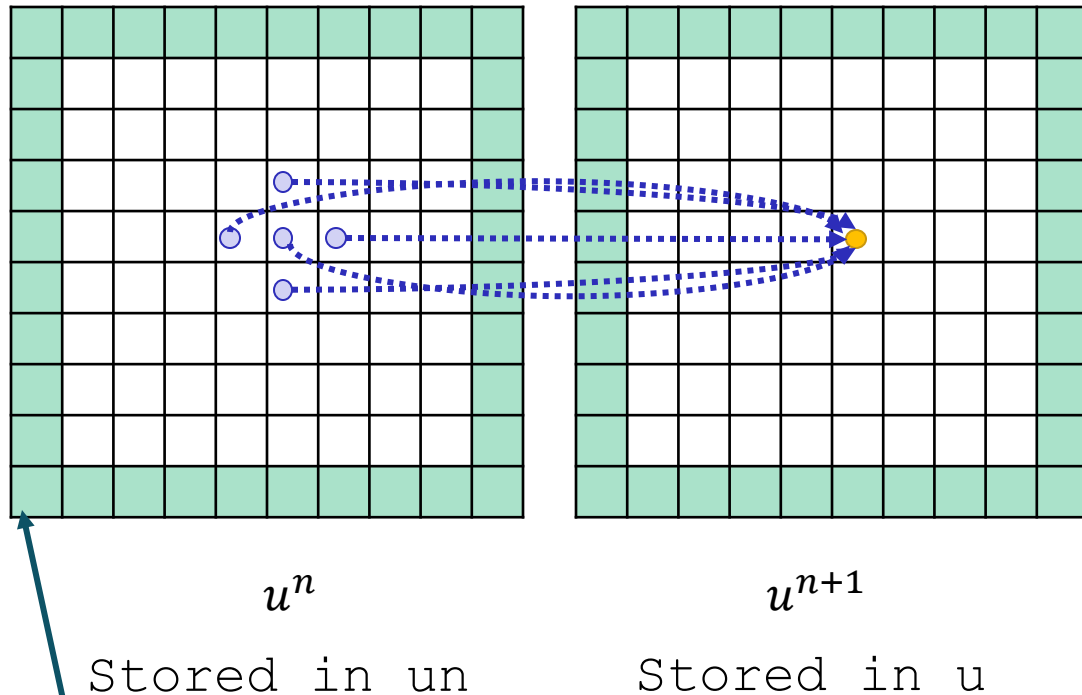
Fixed boundary values

- This computation uses a "5 point stencil"
- Each point in 'u' can be computed in parallel
  - this is accomplished using a 'forall' loop

```
7    ...
8      forall (i, j) in omegaHat do
9        u[i, j] = un[i, j] + alpha * (
10                   un[i-1, j] + un[i, j-1] +
11                   un[i+1, j] + un[i, j+1] -
12                   4 * un[i, j]);
13     ...
```

$$u_{i,j}^{n+1} = u_{i,j}^n + \alpha\left(u_{i-1,j}^n + u_{i,j-1}^n + u_{i+1,j}^n + u_{i,j+1}^n - 4u_{i,j}^n\right)$$

BLOCK DISTRIBUTED & PARALLEL 2D HEAT EQUATION
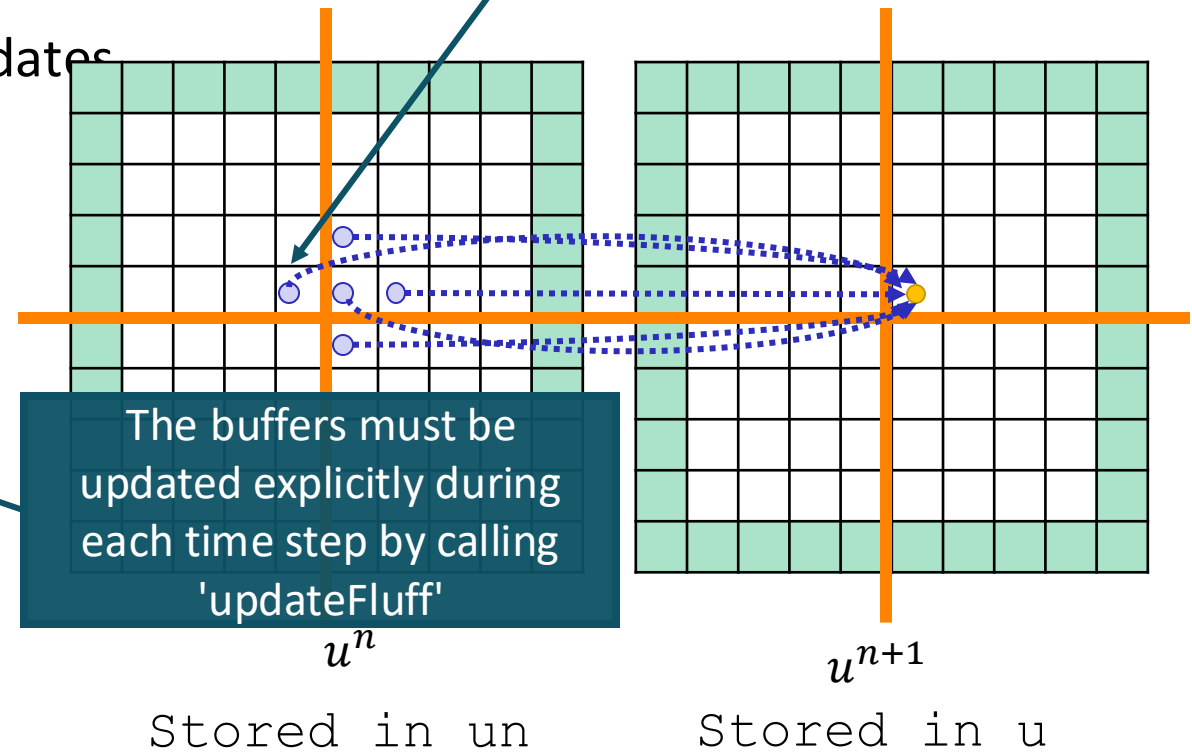
• Declaring distributed domains with the block distribution

```
const Omega = blockDist.createDomain(0..<nx, 0..<ny),
      OmegaHat = Omega.expand(-1);
```
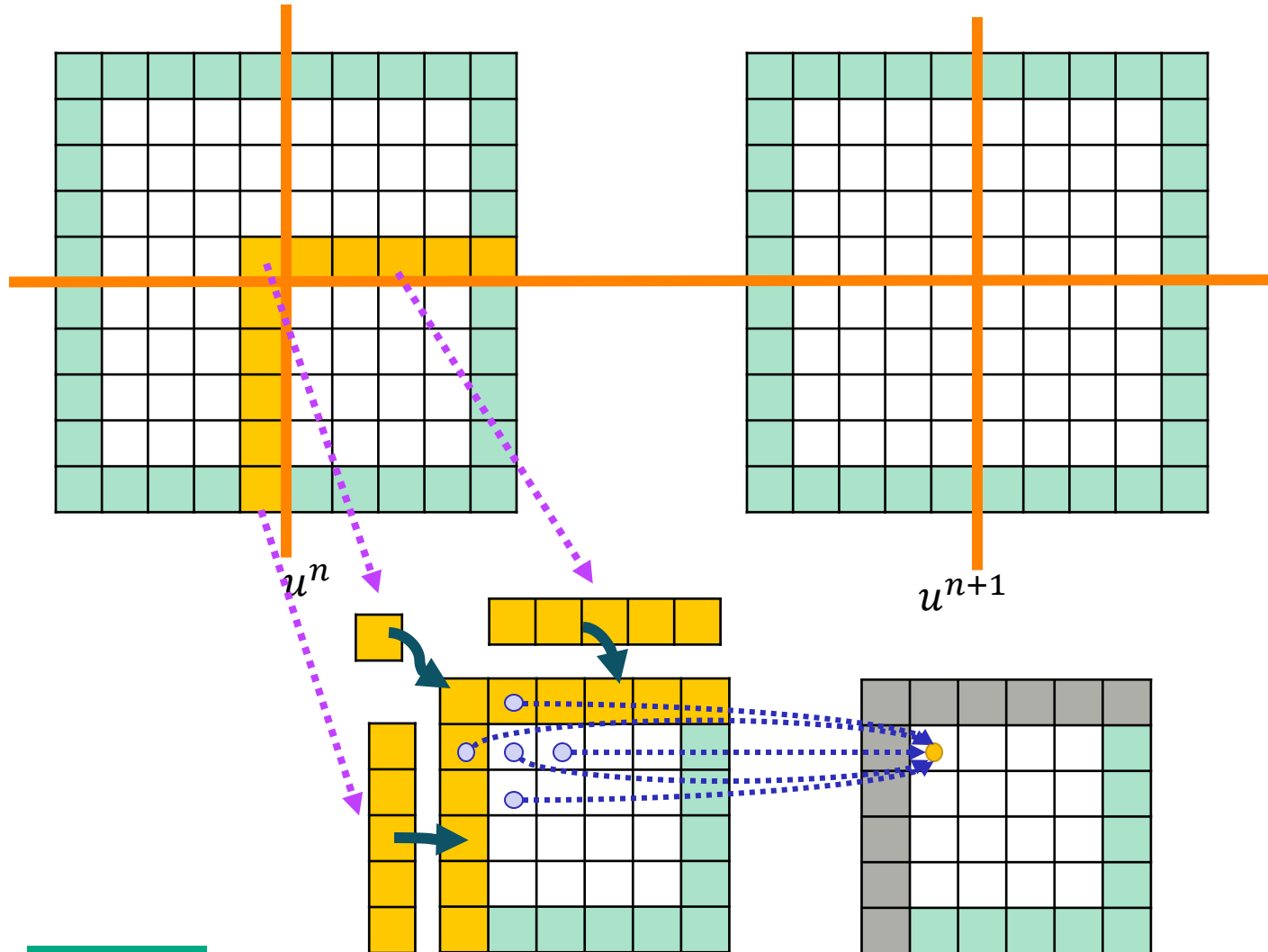
• Distributed & Parallel loop over 'OmegaHat'

```
for 1..nt {
  u <=> un;

  forall (i, j) in OmegaHat do
    u[i, j] = un[i, j] + alpha * (
            un[i-1, j] + un[i, j-1] +
            un[i+1, j] + un[i, j+1] -
            4 * un[i, j]);

}
```

Array access across locale boundaries automatically invokes communication

u^n

Stored in un

u^{n+1}

Stored in u
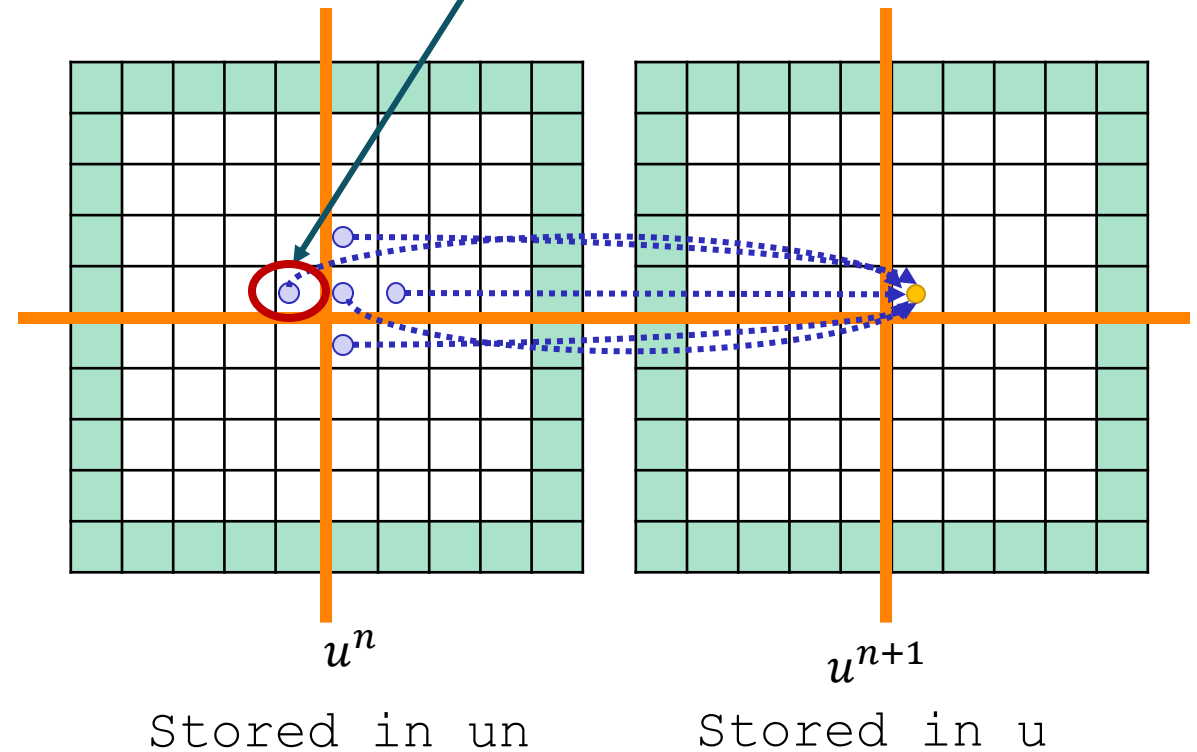
STENCIL DISTRIBUTED & PARALLEL 2D HEAT EQUATION

- Declaring distributed domains with the stencil distribution

```
const Omega = stencilDist.createDomain(
                {0..<nx, 0..<ny}, fluff=(1,1)),
      OmegaHat = Omega.expand(-1);
```

- Distributed & Parallel loop including buffer updates

```
for 1..nt {
  u <=> un;
  un.updateFluff();
  forall (i, j) in OmegaHat do
    u[i, j] = un[i, j] + alpha * (
              un[i-1, j] + un[i, j-1] +
              un[i+1, j] + un[i, j+1] -
              4 * un[i, j]);
}
```

Array access across locale boundaries (within the fluff region) results in a local buffer access — no communication is required

The buffers must be updated explicitly during each time step by calling 'updateFluff'

u^n

u^{n+1}

Stored in un   Stored in u

STENCIL DISTRIBUTED & PARALLEL 2D HEAT EQUATION

u^n

u^{n+1}

- Each locale owns a region of the array surrounded by a "fluff" (buffer) region

- Calling 'updateFluff' copies values from neighboring regions of the array into the local buffered region

- Subsequent accesses of those values result in a local memory access, rather than a remote communication

COMM DIAGNOSTICS

The 'CommDiagnostics' module provides functions for tracking comm between locales
- the following is a common pattern:

```
use CommDiagnostics;
...
startCommDiagnostics();
potentiallyCommHeavyOperation();
stopCommDiagnostics();
...
printCommDiagnosticsTable();
```

- which results in a table summarizing comm counts between the **start** and **stop** calls, e.g.,

locale	get	put	execute_on	execute_on_nb
0	10	0	6	12
1	105	5	0	0
2	105	4	0	0
3	105	7	0	0

- Compiling with '--no-cache-remote' before collecting comm diagnostics is recommended

HANDS ON: HEAT 2D COMM DIAGNOSTICS RESULTS

- Comparing comm diagnostics for:
  - heat-2D-block.chpl
  - heat-2D-stencil.chpl
- *Compilation:*

```
chpl heat-2D-block.chpl --fast
    --no-cache-remote -sRunCommDiag=true

chpl heat-2D-stencil.chpl --fast
    --no-cache-remote -sRunCommDiag=true
```

- *Execution:*

```
./heat-2D-block -nl4 --nx=256 --ny=256

./heat-2D-stencil -nl4 --nx=512 --ny=512
```

- **Block:** number of gets scales with size
- **Stencil:** static number of gets per iteration

Number of Gets on 4 Locales – Block vs. Stencil

number of remote gets doubles with quadrupled problem size

Legend: Stencil (256x256), Stencil (512x512), Block (256x256), Block (512x512)
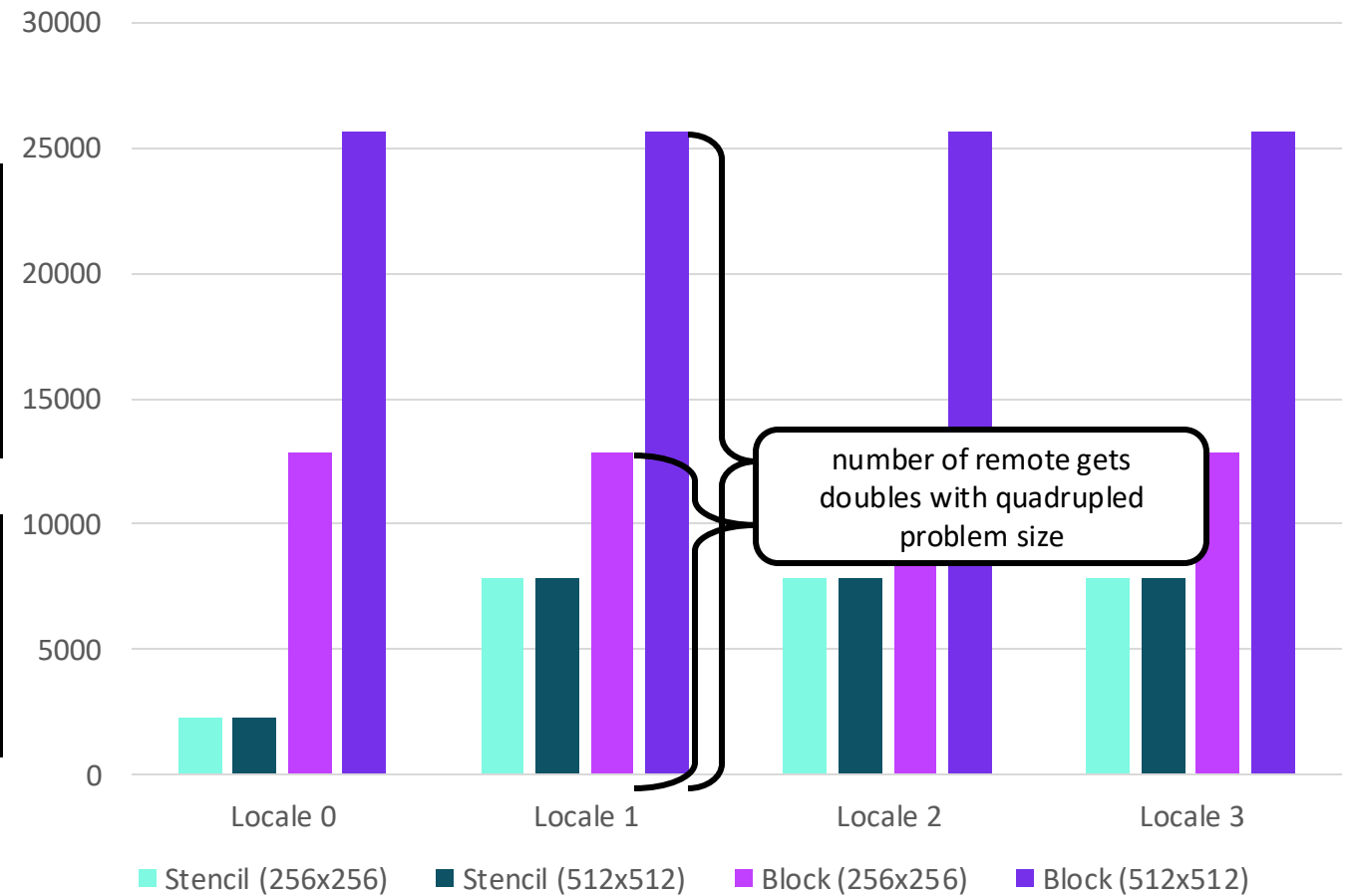
SUMMARIZING WHAT WE LEARNED ABOUT PARALLELISM IN CHAPEL

- Data parallelism session
  - Provides shared memory and distributed memory parallelism
  - Distributions like block and cyclic can be applied to arrays of any dimension
  - Main control abstraction is the 'forall' loop
  - 'forall' loop uses default iterator over provided array or domain, but can use own iterator
    - This is an example of multi-resolution design in Chapel, i.e., the 'forall' loop is mapped down to lower-level abstractions like 'coforall'
  - CommDiagnostics module can be used to observe the number of remote puts/writes and gets/reads at runtime

CHAPEL RESOURCES
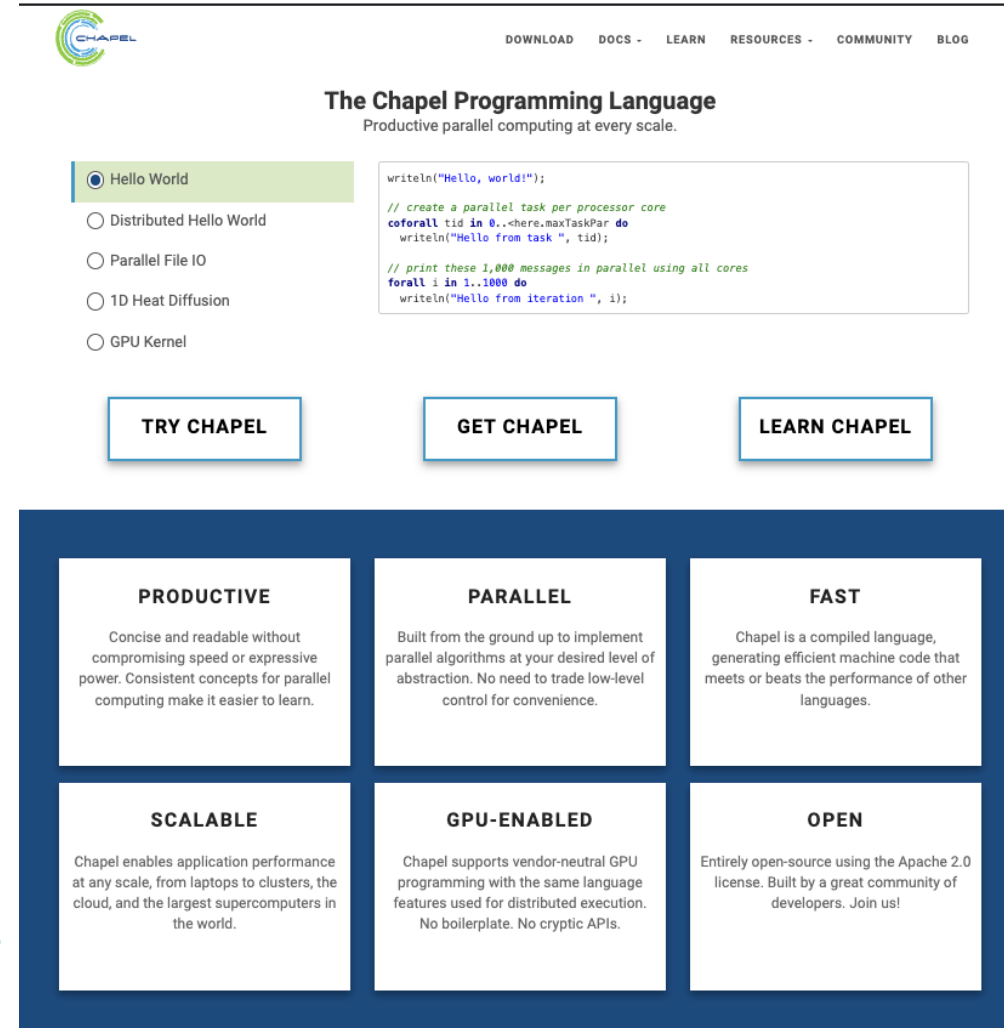
Chapel homepage: https://chapel-lang.org
- (points to all other resources)

Social Media:
- Twitter: @ChapelLanguage
- Facebook: @ChapelLanguage
- YouTube: http://www.youtube.com/c/ChapelParallelProgrammingLanguage

Community Discussion / Support:
- Discord: https://discord.com/invite/xu2xg45yqH
- Stack Overflow: https://stackoverflow.com/questions/tagged/chapel
- GitHub Issues: https://github.com/chapel-lang/chapel/issues