

Chapel: Locality and Affinity

Steve Deitz

Cray Inc.

Outline

- Multi-Locale Basics
 - Locales
 - On, here, and communication
- Distributed Domains and Arrays

The Locale Type

- Definition
 - Abstract unit of target architecture
 - Capacity for processing and storage
 - Supports reasoning about locality
- Properties
 - Locale's tasks have uniform access to local memory
 - Other locale's memory is accessible, but at a price
- Examples
 - A multi-core processor
 - An SMP node

Program Startup

- Execution Context

```

config const numLocales: int;
const LocaleSpace: domain(1) = [0..numLocales-1];
const Locales: [LocaleSpace] locale;
  
```

- Specify # of locales when running executable

```

prompt> a.out --numLocales=8
  
```

Alternatively,

```

prompt> a.out -nl 8
  
```

numLocales: 8

LocaleSpace:

--	--	--	--	--	--	--	--

Locales:

L0	L1	L2	L3	L4	L5	L6	L7
----	----	----	----	----	----	----	----

Rearranging Locales

Create locale views with standard array operations:

```

var TaskALocs = Locales[0..1];
var TaskBLocs = Locales[2..numLocales-1];

var Grid2D = Locales.reshape([1..2, 1..4]);
  
```

Locales: L0 L1 L2 L3 L4 L5 L6 L7

TaskALocs: L0 L1

TaskBLocs: L2 L3 L4 L5 L6 L7

Grid2D:

L0	L1	L2	L3
L4	L5	L6	L7

Locale Methods

- `def locale.id: int { ... }`

Returns index in LocaleSpace

- `def locale.name: string { ... }`

Returns name of locale (like `uname -a`)

- `def locale.numCores: int { ... }`

Returns number of cores available to locale

- `def locale.physicalMemory(...) { ... }`

Returns physical memory available to locale

Example

```
const totalSystemMemory =
  + reduce Locales.physicalMemory();
```

The On Statement

- Syntax

```

on-stmt:
  on expr { stmt }
  
```

- Semantics

- Executes *stmt* on the locale specified by *expr*
- Does not introduce concurrency

- Example

```

var A: [LocaleSpace] int;
coforall loc in Locales do on loc do
  A(loc.id) = compute(loc.id);
  
```

Querying a Variable's Locale

- Syntax

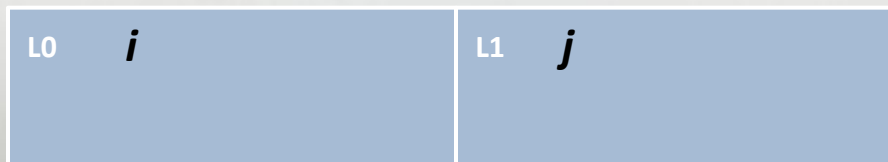
```
locale-query-expr:
  expr . locale
```

- Semantics

- Returns the locale on which *expr* is allocated

- Example

```
var i: int;
on Locales(1) {
  var j: int;
  writeln(i.locale.id, j.locale.id); // outputs 01
}
```



Here

- Built-in locale

```
const here: locale;
```

- Semantics

- Refers to the locale on which the task is executing

- Example

```
writeln(here.id); // outputs 0
on Locales(1) do
    writeln(here.id); // outputs 1
```

Serial Example with Implicit Communication

```

var x, y: real;           // x and y allocated on locale 0

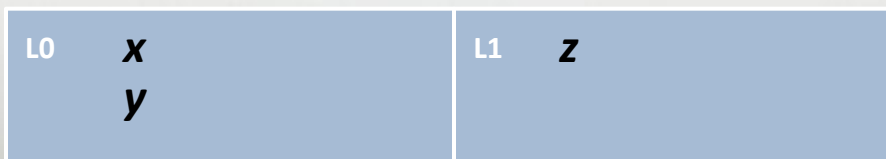
on Locales(1) {           // migrate task to locale 1
    var z: real;         // z allocated on locale 1

    z = x + y;             // remote reads of x and y

    on Locales(0) do      // migrate back to locale 0
        z = x + y;         // remote write to z
                            // migrate back to locale 1

    on x do               // data-driven migration to locale 0
        z = x + y;         // remote write to z
                            // migrate back to locale 1
}                            // migrate back to locale 0

```



The Fragmented Model in Chapel

```

def main() {
    coforall loc in Locales do on loc {
        myFragmentedMain();
    }
}

def myFragmentedMain() {
    const size = numLocales, rank = here.id;
    ...
}

```

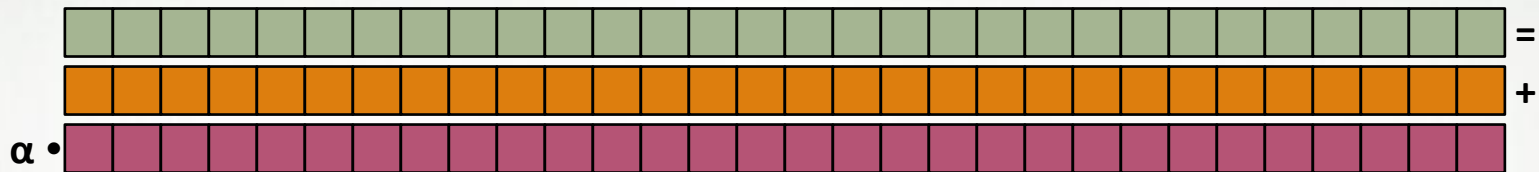
Outline

- Multi-Locale Basics
- Distributed Domains and Arrays

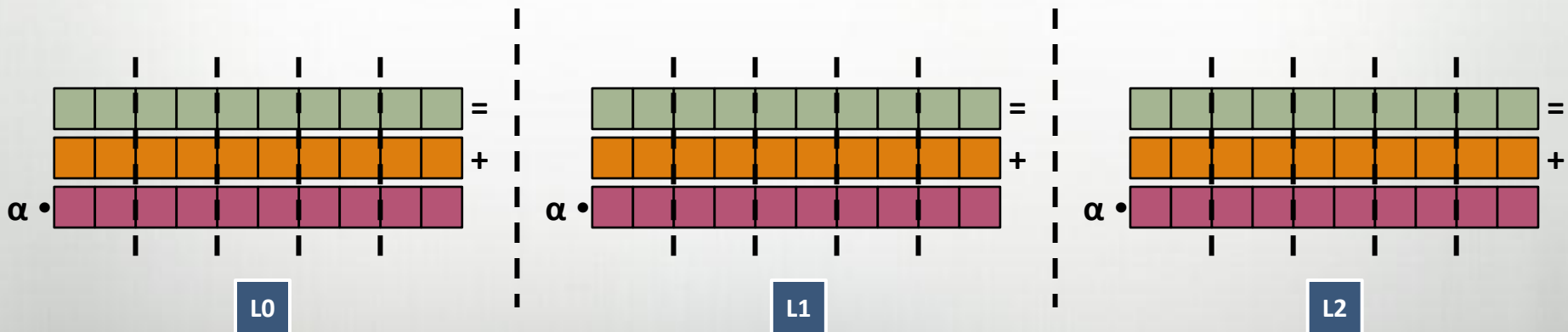
What is a Distribution?

A “recipe” for distributed arrays that...

Instructs the compiler how to Map the global view...



...to a fragmented, per-processor implementation

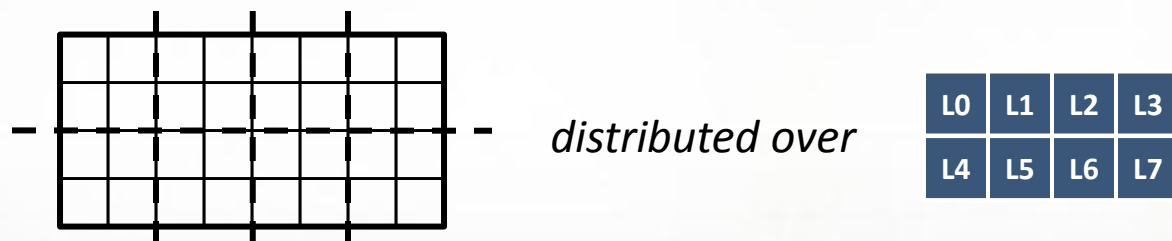


Distributing a Domain

Domains are associated to a distribution

```

const Dist = new Block(rank=2, bbox=[1..4, 1..8]);
var Dom: domain(2) distributed Dist = [1..4, 1..8];
  
```



The distribution defines:

- Ownership of domain indices and array elements
- Default distribution of work (task-to-locale map)
E.g., forall loops over distributed domains/arrays

Authoring Distributions

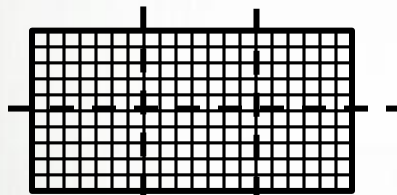
- (Advanced) programmers can write distributions
- Built-in library of distributions
 - No extra compiler support for built-in distributions
 - Compiler uses structural interface:
 - Create domains and arrays
 - Map indices to locales
 - Access array elements
 - Iterate over indices/elements sequentially, in parallel, zippered
 - ...
- Distributions are built using language-level concepts
 - On for data and task locality
 - Begin, cobegin, and coforall for data parallelism

Distributing Domains

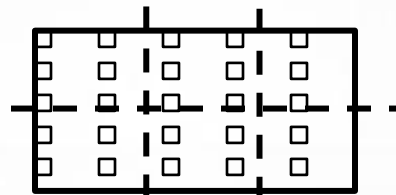
All domain types can be distributed.

Semantics are independent of distribution.

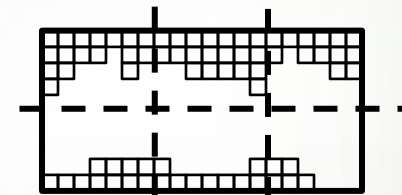
(Though performance and parallelism will vary...)



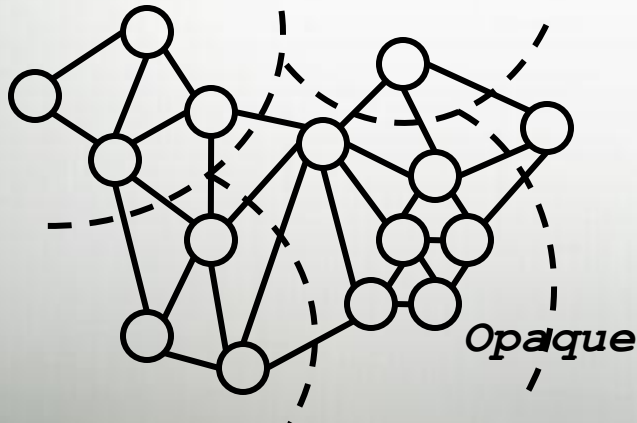
Dense



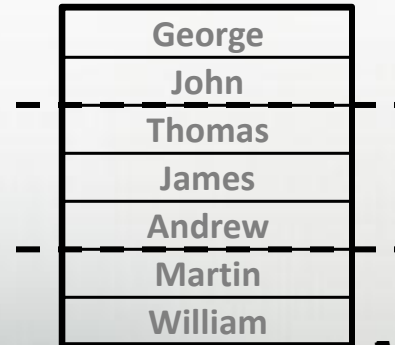
Strided



Sparse



Opaque



Associative

Questions?

- Multi-Locale Basics
 - Locales
 - On, here, and communication
- Domain and Array Distributions