Chapel: Data Parallelism

Steve Deitz

Cray Inc.

Outline

- Domains and Arrays
  - Overview
  - Arithmetic
- Other Domain Types
- Data Parallel Operations

Domains

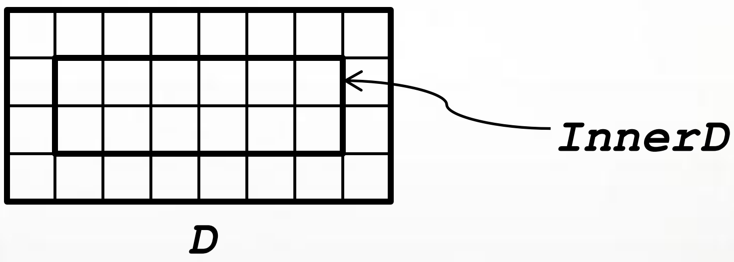
- A first-class index set
  - Specifies size and shape of arrays
  - Supports iteration, array operations
  - Potentially distributed across machines
- Three main classes
  - Arithmetic—indices are Cartesian tuples
  - Associative—indices are hash keys
  - Opaque—indices are anonymous
- Fundamental Chapel concept for data parallelism
- A generalization of ZPL's region concept

Sample Arithmetic Domains

```
config const m = 4, n = 8;

var D: domain(2) = [1..m, 1..n];

var InnerD: domain(2) = [2..m-1, 2..n-1];
```



Domains Define Arrays

- Syntax

```
array-type:
  [ domain-expr ] type
```

- Semantics
  - Associates data with each index in *domain-expr*
- Example

```
var A, B: [D] real;
```

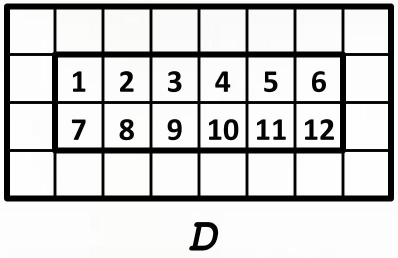
- Revisited example

```
var A: [1..3] int; // creates anonymous domain [1..3]
```

Domain Iteration

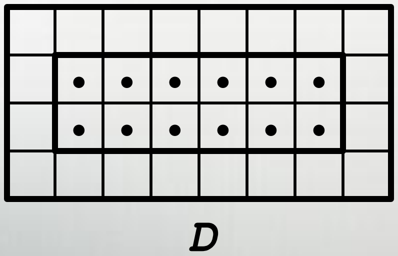
- For loops (discussed already)
  - Executes loop body once per loop iteration
  - Order is serial

```
for i in InnerD do ...
```



- Forall loops
  - Executes loop body once per loop iteration
  - Order is parallel (must be *serializable*)

```
forall i in InnerD do ...
```



Other Forall Loops

Forall loops also support...

- A symbolic shorthand:

```
[(i,j) in D] A(i,j) = i + j/10.0;
```

1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8
3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8
4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8

A

- An expression-based form:

```
A = forall (i,j) in D do i + j/10.0;
```

- A sugar for array initialization:

```
var A: [(i,j) in D] real = i + j/10.0;
```

Usage of For, Forall, and Coforall

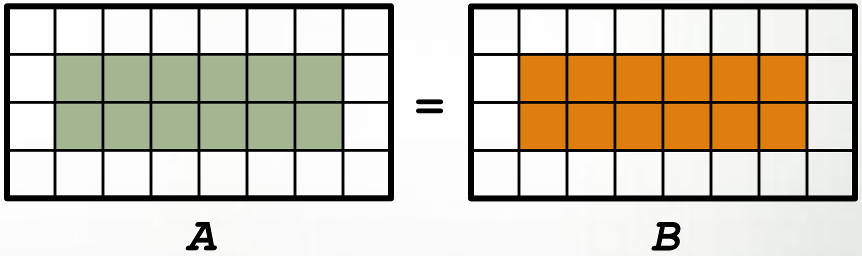
- Use for when
  - A loop must be executed serially
  - One task is sufficient for performance
- Use forall when
  - The loop can be executed in parallel
  - The loop can be executed serially
- Use coforall when
  - The loop must be executed in parallel

    (And not just for performance reasons!)

Other Domain Functionality

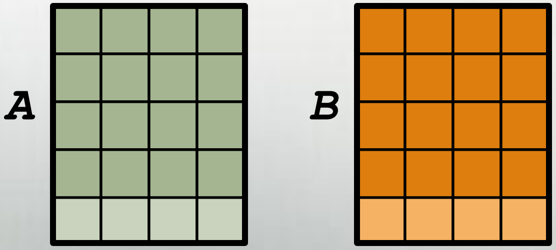
- Domain methods (exterior, interior, translate, ...)
- Domain slicing (intersection)
- Array slicing (sub-array references)

```
A(InnerD) = B(InnerD);
```



- Array reallocation
  - Reassign domain → change array
  - Values are preserved (new elements initialized)

```
D = [1..m+1, 1..m];
```

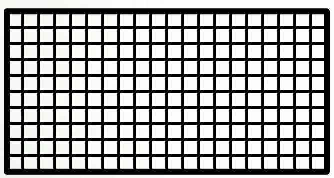


Outline

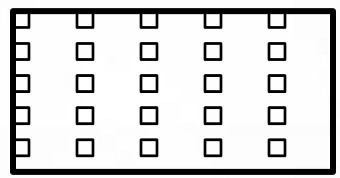
- Domains and Arrays
- Other Domain Types
  - Strided
  - Sparse
  - Associative
  - Opaque
- Data Parallel Operations

The Varied Kinds of Domains

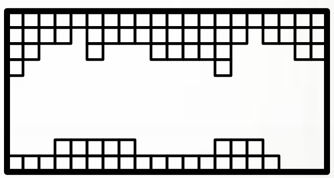
```
var Dense: domain(2) = [1..10, 1..20],
    Strided: domain(2) = Dense by (2, 4),
    Sparse: subdomain(Dense) = genIndices(),
    Associative: domain(string) = readNames(),
    Opaque: domain(opaque);
```



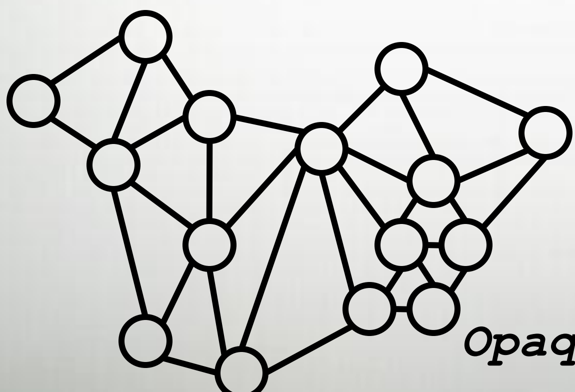
Dense



Strided



Sparse



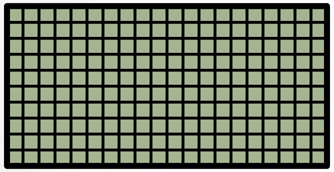
Opaque

George
John
Thomas
James
Andrew
Martin
William

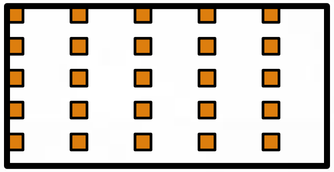
Associative

The Varied Kinds of Arrays

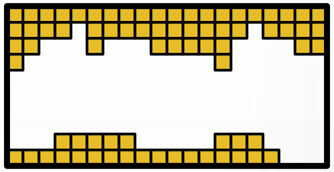
```
var DenseArr: [Dense] real,
    StridedArr: [Strided] real,
    SparseArr: [Sparse] real,
    AssociativeArr: [Associative] real,
    OpaqueArr: [Opaque] real;
```



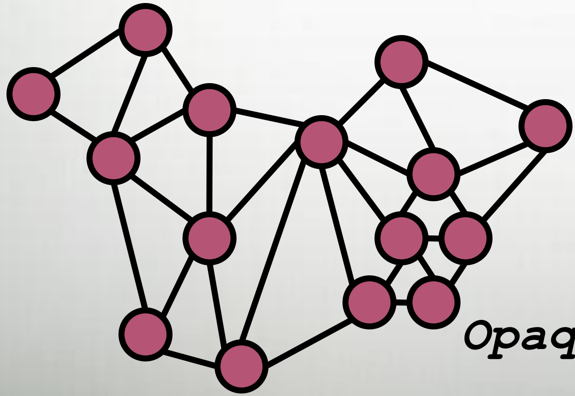
DenseArr



StridedArr



SparseArr

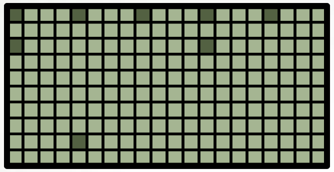


OpaqueArr

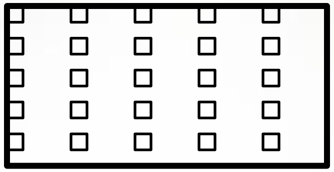
AssociativeArr

All Domains Support Iteration

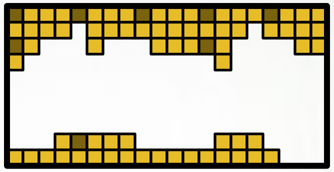
```
forall (i,j) in Strided {
  DenseArr(i,j) += SparseArr(i,j);
}
```



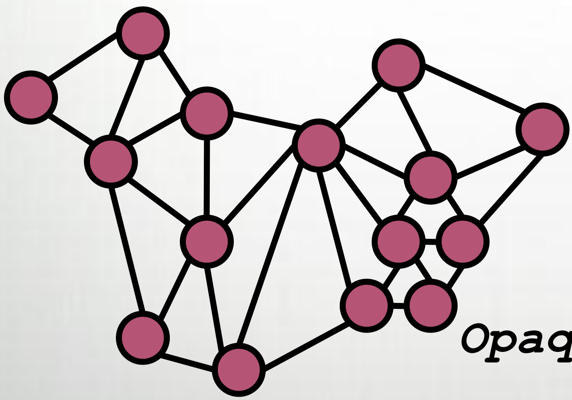
DenseArr



Strided



SparseArr



OpaqueArr

AssociativeArr

(Also, all domains support slicing, reallocation, ...)

Associative Domains and Arrays by Example

```
var Presidents: domain(string) =
      (“George”, “John”, “Thomas”,
       “James”, “Andrew”, “Martin”);

Presidents += “William”;


var Ages: [Presidents] int,
    Birthdays: [Presidents] string;

Birthdays(“George”) = “Feb 22”;


forall president in Presidents do
  if Birthdays(president) == today then
    Ages(president) += 1;
```

George
John
Thomas
James
Andrew
Martin
William

Presidents

Feb 22	277
Oct 30	274
Apr 13	266
Mar 16	251
Mar 15	242
Dec 5	227
Feb 9	236

Birthdays ***Ages***

Outline

- Domains and Arrays
- Other Domain Types
- Data Parallel Operations
  - Promotion
  - Reductions and scans

Data Parallel Promotion

Functions/operators expecting scalars can also take...

- Arrays, causing each element to be passed

```
...sin(A)...
...2*A...
```

≈

```
...[a in A] sin(a)...
...[a in A] 2*a...
```

- Domains, causing each index to be passed

```
foo(Sparse); // calls foo for all indices in Sparse
```

Multiple arguments can promote using either...

- Zipper promotion

```
...pow(A, B)...
```

≈

```
...[(a,b) in (A,B)] pow(a,b)...
```

- Tensor promotion

```
...pow[A, B]...
```

≈

```
...[(a,b) in [A,B]] pow(a,b)...
```

Reductions

- Syntax

```
reduce-expr:
  reduce-op reduce iterator-expr
```

- Semantics
  - Combines iterated elements with *reduce-op*
  - *Reduce-op* may be built-in or user-defined
- Examples

```
total = + reduce A;
bigDiff = max reduce [i in InnerD] abs(A(i)-B(i);
```

Scans

- Syntax

```
scan-expr:
  scan-op scan iterator-expr
```

- Semantics
  - Computes parallel prefix of *scan-op* over elements
  - *Scan-op* may be any *reduce-op*
- Examples

```
var A, B, C: [1..5] int;
A = 1;                        // A:  1  1  1  1  1
B = + scan A;                 // B:  1  2  3  4  5
B(3) = -B(3);                 // B:  1  2 -3  4  5
C = min scan B;               // C:  1  1 -3 -3 -3
```

Reduction and Scan Operators

- Built-in
  - +, *, &&, ||, &, |, ^, min, max
  - minloc, maxloc

    (Generate a tuple of the min/max and its index)
- User-defined
  - Defined via a class that supplies a set of methods
  - Compiler generates code that calls these methods
  - More information:

    S. J. Deitz, D. Callahan, B. L. Chamberlain, and L. Snyder. *Global-view abstractions for user-defined reductions and scans*. In Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, 2006.

Questions?

- Domains and Arrays
  - Overview
  - Arithmetic
- Other Domain Types
  - Strided
  - Sparse
  - Associative
  - Opaque
- Data Parallel Operations
  - Promotion
  - Reductions and scans