Chapel: Data Parallelism
Outline

- Domains and Arrays
  - Overview
  - Arithmetic
- Other Domain Types
- Data Parallel Operations
- NAS MG Stencil Revisited
Domains

- A first-class index set
  - Specifies size and shape of arrays
  - Supports iteration, array operations
  - Potentially distributed across locales
- 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
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**
  
  \[
  \text{array-type:} \quad [ \text{domain-expr} ] \text{ elt-type}
  \]

- **Semantics**
  - Stores element for each index in `domain-expr`

- **Example**

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

- **Revisited example**

  ```chapel
  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 ...
```
Forall loops also support...

- **A shorthand:**
  
  \[
  ([i,j] \text{ in } D] A(i,j) = i + j/10.0;
  \]

- **An expression-based form:**
  
  \[
  A = \text{forall } (i,j) \text{ in } D \text{ do } i + j/10.0;
  \]

- **A shorthand expression-based form:**
  
  \[
  A = [(i,j) \text{ in } D] i + j/10.0;
  \]
Data Parallelism Configuration Constants

- **--dataParTasksPerLocale=#**
  - Specify # of tasks to execute forall loops
  - Default: number of cores (*in current implementation*)

- **--dataParIgnoreRunningTasks=[true | false]**
  - If false, reduce # of forall tasks by # of running tasks
  - Default: true (*in current implementation*)

- **--dataParMinGranularity=#**
  - If > 0, reduce # of forall tasks if any task has fewer iterations
  - Default: 1 (*in current implementation*)
Other Domain Functionality

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

\[
A(\text{InnerD}) = B(\text{InnerD});
\]

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

\[
D = [1..m+1, 1..m];
\]
Arrays are passed by reference

```chapel
def f(A: []) { A = 0; }

f(A[InnerD]);
```

Non-argument array alias of a slice

```chapel
var AA => A(InnerD);
```

Re-indexing arrays

```chapel
def f(A: [1..n-2,1..m-2]);

f(A[2..n-1,2..m-1]);

var AA: [1..n-2,1..m-2] => A[2..n-1,2..m-1];
```
Outline

- Domains and Arrays
- Other Domain Types
  - Strided
  - Sparse
  - Associative
  - Opaque
- Data Parallel Operations
- NAS MG Stencil Revisited
The Varied Kinds of Domains

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

\begin{verbatim}
var DenseArr: [Dense] real,
StridedArr: [Strided] real,
SparseArr: [Sparse] real,
AssociativeArr: [Associative] real,
OpaqueArr: [Opaque] real;
\end{verbatim}
```chapel
forall (i,j) in Strided {
    DenseArr(i,j) += SparseArr(i,j);
}
```

(Also, all domains support slicing, reallocation, ...)
**Associative Domains and Arrays by Example**

```chapel
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;
```

**Presidents**
- George
- John
- Thomas
- James
- Andrew
- Martin
- William

**Birthdays**
- Feb 22
- Oct 30
- Apr 13
- Mar 16
- Mar 15
- Dec 5
- Feb 9

**Ages**
- 277
- 274
- 266
- 251
- 242
- 227
- 236
More Domain and Array Examples

- examples/primers/slices.chpl
- examples/primers/sparse.chpl
- examples/primers/opaque.chpl
Outline

- Domains and Arrays
- Other Domain Types
- Data Parallel Operations
  - Promotion
  - Reductions
  - Scans
- NAS MG Stencil Revisited
Functions/operators expecting scalars can also take...

- **Arrays**, causing each element to be passed

  \[
  \sin(A) \\
  2*A
  \]

  \[
  \approx \\
  \text{forall } a \text{ in } A \text{ do } \sin(a) \\
  \text{forall } a \text{ in } A \text{ do } 2*a
  \]

- **Domains**, causing each index to be passed

  \[
  \text{foo}(\text{Sparse})
  \]

  \[
  \approx \\
  \text{forall } i \text{ in } \text{Sparse} \text{ do } \text{foo}(i)
  \]

Multiple arguments can promote using either...

- **Zipper promotion**

  \[
  \text{pow}(A, B)
  \]

  \[
  \approx \\
  \text{forall } (a,b) \text{ in } (A,B) \text{ do } \text{pow}(a,b)
  \]

- **Tensor product promotion**

  \[
  \text{pow}[A, B]
  \]

  \[
  \approx \\
  \text{forall } (a,b) \text{ in } [A,B] \text{ do } \text{pow}(a,b)
  \]
Reductions

- **Syntax**

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

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

- **Examples**

  ```plaintext
  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

```chapel
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:

Reduction Examples

- examples/primers/reductions.chpl
Outline

- Domains and Arrays
- Other Domain Types
- Data Parallel Operations
- NAS MG Stencil Revisited
NAS MG Stencil Revisited

\[ = w_0 + w_1 + w_2 + w_3 \]
def rprj3(S, R) {
    const Stencil = [-1..1, -1..1, -1..1],
        W: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
        W3D = [(i,j,k) in Stencil] W((i!=0)+(j!=0)+(k!=0));

    forall inds in S.domain do
        S(inds) =
            + reduce [offset in Stencil] (W3D(offset) *
                R(inds + offset*R.stride));
}

•Domains and Arrays
  • Overview
  • Arithmetic
•Other Domain Types
  • Strided
  • Sparse
  • Associative
  • Opaque
•Data Parallel Operations
  • Promotion
  • Reductions
  • Scans
•NAS MG stencil revisited