Chapel: Domain Maps
(Layouts and Distributions)
### Multi-locale Data Parallel Hello World

```chapel
cfg const numIters = 100000;
const WorkSpace = [1..numIters] dmapped Block(...);

forall i in WorkSpace do
    writeln("Hello, world! ",
            "from iteration ", i, " of ", numIters,
            " on locale ", here.id, " of ", numLocales);
```
Domains are first-class index sets
- Specify the size and shape of arrays
- Support iteration, array operations, etc.
Q1: How are arrays laid out in memory?
   - Are regular arrays laid out in row- or column-major order? Or…?
   - What data structure is used to store sparse arrays? (COO, CSR, ...?)

Q2: How are data parallel operators implemented?
   - How many tasks?
   - How is the iteration space divided between the tasks?

* ...?
Q3: How are arrays distributed between locales?
- Completely local to one locale? Or distributed?
- If distributed... In a blocked manner? cyclically? block-cyclically?
  recursively bisected? dynamically rebalanced? ...?

Q4: What architectural features will be used?
- Can/Will the computation be executed using CPUs? GPUs? both?
- What memory type(s) is the array stored in? CPU? GPU? texture? ...

A1: In Chapel, any of these could be the correct answer
A2: Chapel’s *domain maps* are designed to give the user full control over such decisions
Outline

- Data Parallelism Revisited
- Domain Maps
  - Layouts
  - Distributions
Domain maps are “recipes” that instruct the compiler how to map the global view of a computation...

\[ A = B + \alpha \cdot C; \]

...to the target locales’ memory and processors:
Domain Maps

Domain Maps: “recipes for implementing parallel/distributed arrays and domains”

They define data storage:
- Mapping of domain indices and array elements to locales
- Layout of arrays and index sets in each locale’s memory

...as well as operations:
- random access, iteration, slicing, reindexing, rank change, ...
- the Chapel compiler generates calls to these methods to implement the user’s array operations
Domain Maps fall into two major categories:

**layouts:** target a single shared memory segment
- (that is, a desktop machine or multicore node)
- **examples:** row- and column-major order, tilings, compressed sparse row

**distributions:** target distinct memory segments
- (that is a distributed memory cluster or supercomputer)
- **examples:** Block, Cyclic, Block-Cyclic, Recursive Bisection, ...
```plaintext
const ProblemSpace = [1..m];

var A, B, C: [ProblemSpace] real;

A = B + alpha * C;
```

No domain map specified => use default layout
• current locale owns all indices and values
• computation will execute using local processors only
STREAM Triad: Chapel (multinode, blocked)

```
const ProblemSpace = [1..m]

dmapped Block(boundingBox=[1..m]);

var A, B, C: [ProblemSpace] real;

A = B + alpha * C;
```
const ProblemSpace = [1..m]

\textbf{dmapped} Cyclic(startIdx=1);

\textbf{var} A, B, C: [ProblemSpace] real;

A = B + alpha * C;
Some Standard Distributions: Block and Cyclic

```
var Dom: domain(2) dmapped Block(boundingBox=[1..4, 1..8]) = [1..4, 1..8];
```

```
var Dom: domain(2) dmapped Cyclic(startIdx=(1,1)) = [1..4, 1..8];
```
All Domain Types Support Domain Maps

- **dense**
- **strided**
- **sparse**
- **unstructured**
- **associative**
Chapel’s Domain Map Philosophy

1. Chapel provides a library of standard domain maps
   • to support common array implementations effortlessly

2. Advanced users can write their own domain maps in Chapel
   • to cope with shortcomings in our standard library

3. Chapel’s standard layouts and distributions will be written using the same user-defined domain map framework
   • to avoid a performance cliff between “built-in” and user-defined domain maps

4. Domain maps should only affect implementation and performance, not semantics
   • to support switching between domain maps effortlessly
• Domain types and literals may be domain mapped:

```chapel
var Dom: domain(...) dmapped myDMap(...)
   = [...] dmapped myDMap(...);
```

• In practice, this tends to be a great place to rely on type inference to avoid repetition:

```chapel
var Dom = [...] dmapped myDMap(...);
```
Declaring Domain Maps

- **Syntax**
  
  \[
  \text{dmap-type:} \\
  \quad \textbf{dmap} (\text{dmap-class}(\ldots)) \\
  \text{dmap-value:} \\
  \quad \textbf{new dmap} (\textbf{new dmap-class}(\ldots))
  \]

- **Semantics**
  - Domain maps can be declared independently of a domain
  - Useful for declaring multiple domains using the same map

- **Examples**

  ```
  use myDMapMod;
  var DMap: \textbf{dmap} (\text{myDMap}(\ldots)) = \textbf{new dmap} (\textbf{new myDMap}(\ldots));
  
  var Dom: \text{domain}(\ldots) \text{ dmapped } DMap;
  var A: [\text{Dom}] \text{ real};
  ```
Outline

- Data Parallelism Revisited
- Domain Maps
- Chapel Standard Layouts and Distributions
  - Block
  - Cyclic
The Block class constructor

```
proc Block(boundingBox: domain,
    targetLocales: [] locale = Locales,
    dataParTasksPerLocale = ...,
    dataParIgnoreRunningTasks = ...,
    dataParMinGranularity = ...,
    param rank = boundingBox.rank,
    type idxType = boundingBox.dim(1).eltType)
```

distributed to

```
L0 L1 L2 L3
L4 L5 L6 L7
```
The Cyclic class constructor

```plaintext
proc Cyclic(startIdx, targetLocales: [], locale = Locales, dataParTasksPerLocale = ..., dataParIgnoreRunningTasks = ..., dataParMinGranularity = ..., param rank: int = inferred from startIdx, type idxType = inferred from startIdx)
```

distributed to

```
L0  L1  L2  L3
L4  L5  L6  L7
```
"Hello World" in Chapel: a Domain-Map Version

- Multi-locale Data Parallel Hello World

```chapel
config const numIters = 100000;
const WorkSpace = [1..numIters] dmapped Block(...);

forall i in WorkSpace do
    writeln("Hello, world! ",
            "from iteration ", i,
            " of ", numIters,
            " on locale ", here.id,
            " of ", numLocales);
```
For More Information on Domain Maps

**HotPAR’10:** *User-Defined Distributions and Layouts in Chapel: Philosophy and Framework*, Chamberlain, Deitz, Iten, Choi; June 2010

**CUG 2011:** *Authoring User-Defined Domain Maps in Chapel*, Chamberlain, Choi, Deitz, Iten, Litvinov; May 2011

**Chapel release:**

- Technical notes detailing domain map interface for programmers:
  
  \$CHPL_HOME/doc/technotes/README.dsi

- Current domain maps:
  
  \$CHPL_HOME/modules/dists/*.chpl
  layouts/*.chpl
  internal/Default*.chpl
Domain Maps: Status

- Full-featured Block, Cyclic, Replicated distributions
- COO and CSR Sparse layouts supported
- Quadratic probing Associative layout supported
- Block-Cyclic, Dimensional, Associative distributions underway
- User-defined domain map interface still evolving
- Memory currently leaked for distributed arrays
Future Directions

- Advanced uses of domain maps:
  - GPU programming
  - Dynamic load balancing
  - Resilient computation
  - *in situ* interoperability
  - Out-of-core computations

- Improved syntax for declared domain maps
Questions?

- Data Parallelism Revisited
- Domain maps
  - Layouts
  - Distributions
- The Chapel Standard Distributions
  - Block Distribution
  - Cyclic Distribution
- User-defined Domain Maps