Five Powerful Chapel Idioms

Steve Deitz, Brad Chamberlain, Sung-Eun Choi,
David Iten, Lee Prokowich

Cray Inc.

What is Chapel?

- A new parallel programming language
  - Under development at Cray Inc.
  - Supported through the DARPA HPCS program
- Availability
  - Version 1.1 release April 15, 2010
  - Open source via BSD license

    http://chapel.cray.com/

    http://sourceforge.net/projects/chapel/

Chapel Productivity Goals

- Improve programmability over current languages
  - Writing parallel codes
  - Reading, changing, porting, tuning, maintaining, ...
- Support performance at least as good as MPI
  - Competitive with MPI on generic clusters
  - Better than MPI on more capable architectures
- Improve portability over current languages
  - As ubiquitous as MPI
  - More portable than OpenMP, UPC, CAF, ...
- Improve robustness via improved semantics
  - Eliminate common error cases
  - Provide better abstractions to help avoid other errors

Outline

- What is Chapel
- The Five Idioms
  - Data distributions
  - Data-parallel loops
  - [Asynchronous] [remote] tasks
  - Nested parallelism
  - [Remote] transactions
- Performance Study

Idiom 1: Data Distributions

```
const D = [1..n, 1..n];           // domain - index set
var A: [D] real;                  // array - data values
const DD = D dmapped X(...);      // distributed domain
var DA: [DD] real;                // distributed array
```

- Syntax

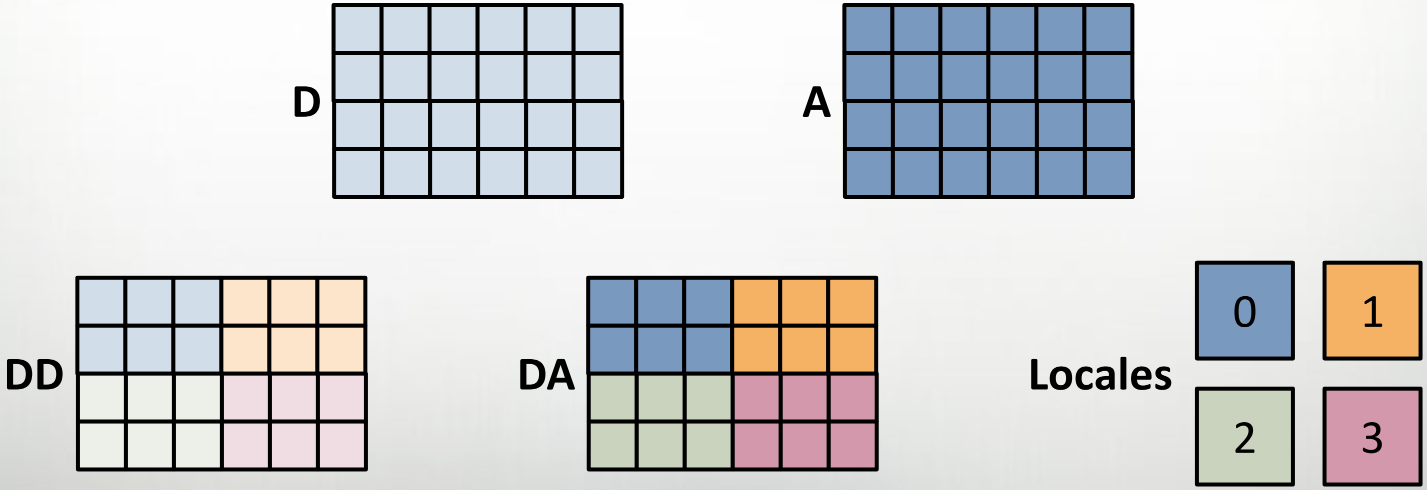
```
domain-expr dmapped distribution-expr
```

- Semantics
  - Index set of *domain-expr* is partitioned via *distribution-expr*
  - Partitioned across 'locales' of a system
  - Locale – abstraction of memory and processing capability

Idiom 1: Data Distributions

- Standard Block distribution

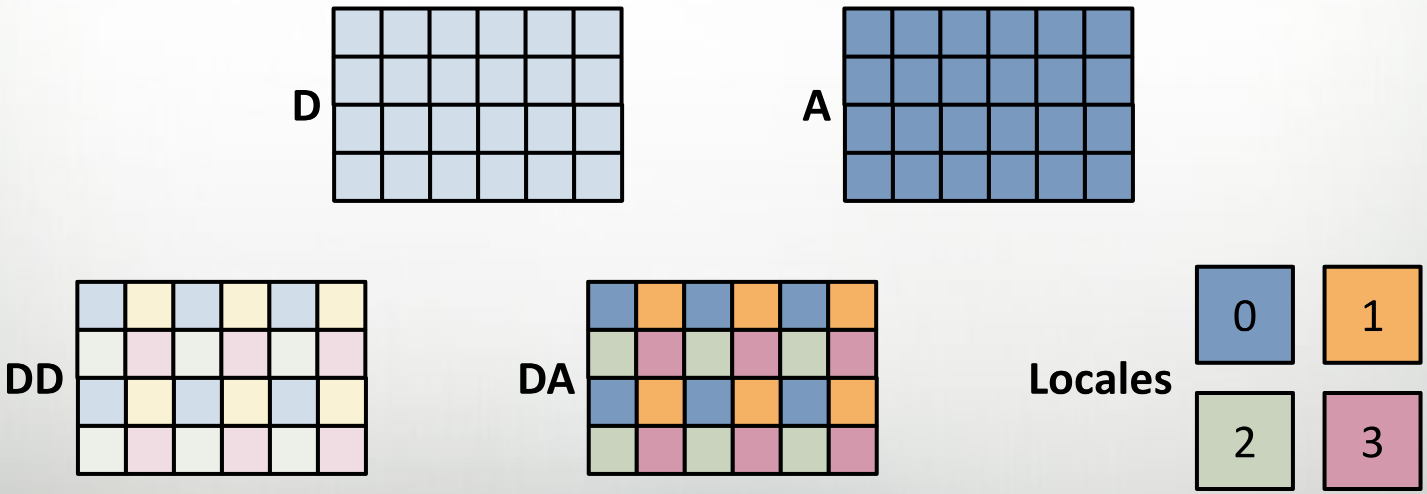
```
const D = [1..n, 1..m];
var A: [D] real;
const DD = D dmapped Block(boundingBox=D);
var DA: [DD] real;
```



Idiom 1: Data Distributions

- Standard Cyclic distribution

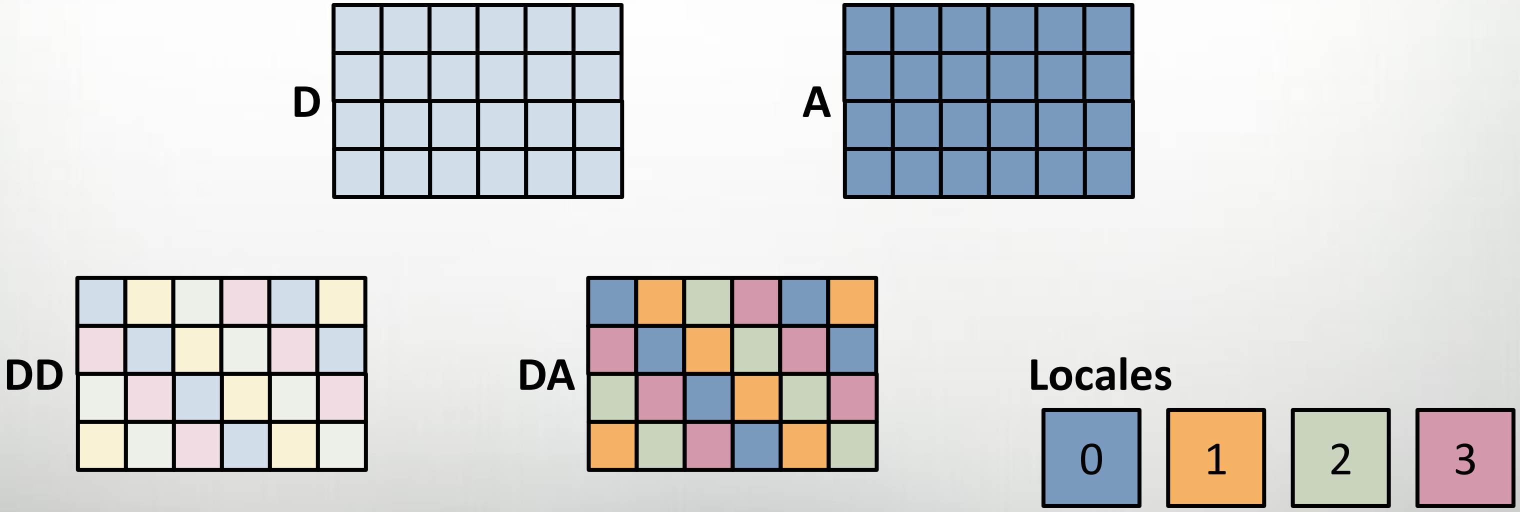
```
const D = [1..n, 1..m];
var A: [D] real;
const DD = D dmapped Cyclic(startIdx=D.low);
var DA: [DD] real;
```



Idiom 1: Data Distributions

- User-defined MyBanded distribution

```
const D = [1..n, 1..m];
var A: [D] real;
const DD = D dmapped MyBanded(startIdx=D.low);
var DA: [DD] real;
```



Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

- Syntax

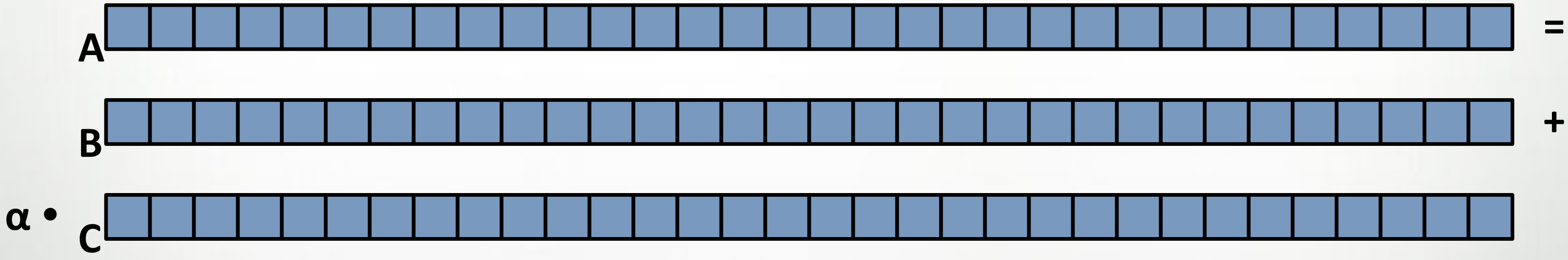
```
forall ( index-exprs ) in ( iterable-exprs ) do
  loop-body-stmts
```

- Semantics
  - Zipped (element-wise) iteration
  - Shapes of iterable expressions must match

Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

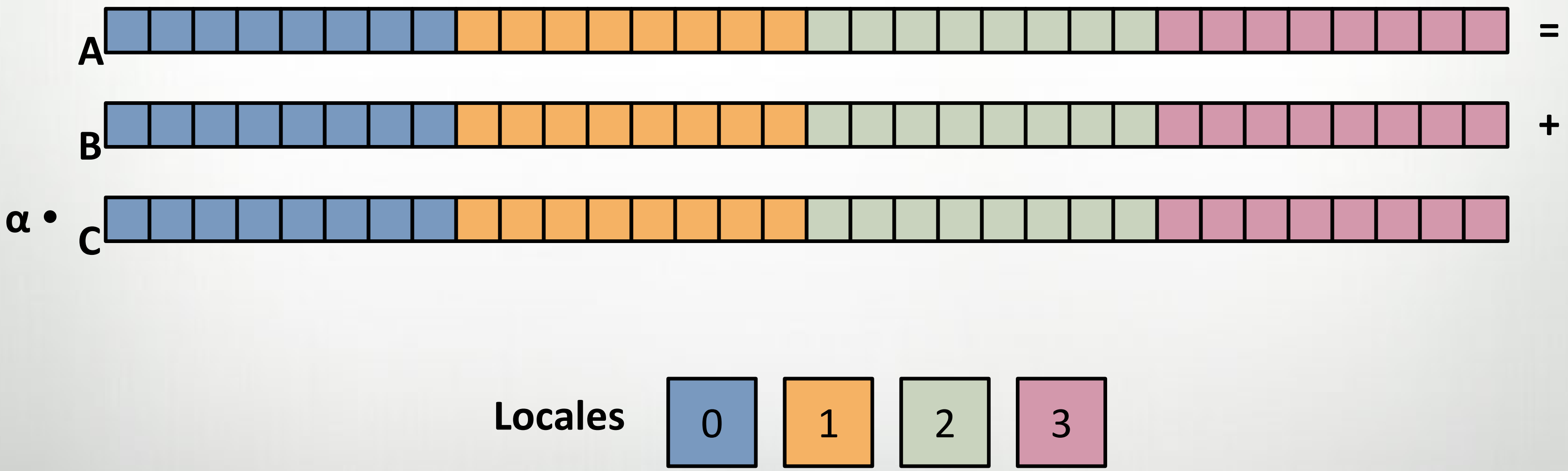
- Example 1: Non-distributed arrays



Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

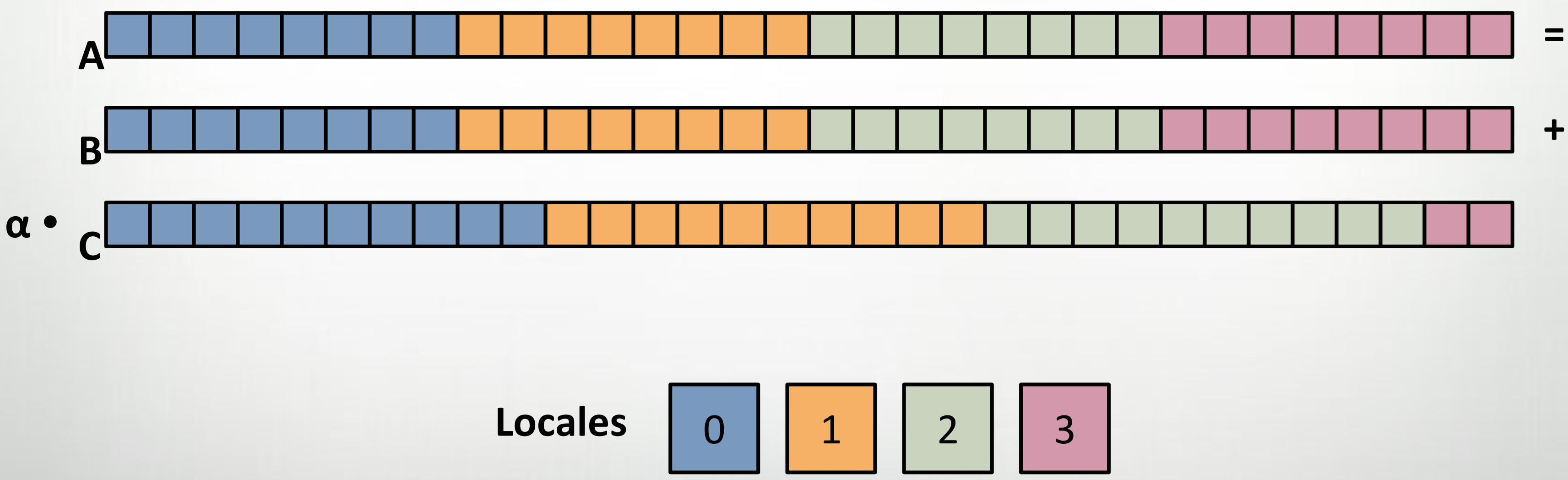
- Example 2: Block-distributed arrays



Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

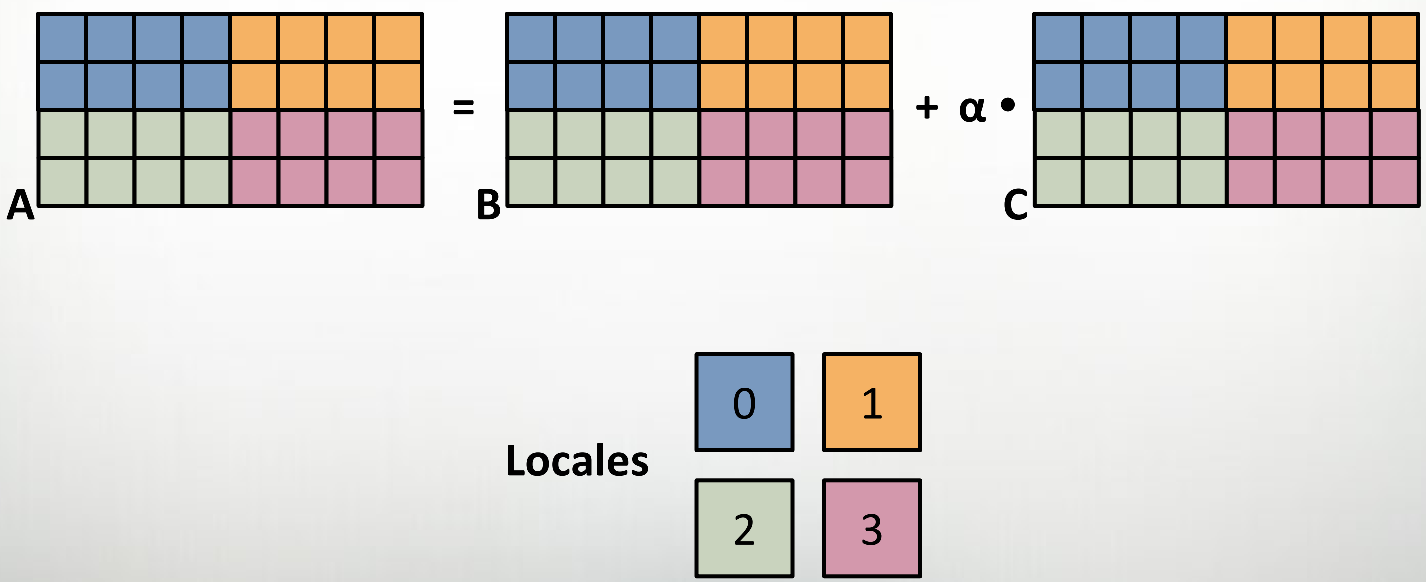
- Example 3: Unaligned block-distributed arrays



Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

- Example 4: 2D Block-distributed arrays



Idiom 2: Data-Parallel Loops

```
forall (a, b, c) in (A, B, C) do
  a = b + alpha * c;
```

- Other possibilities
  - Associative, sparse, and unstructured arrays
  - Domains and iterators with no associated data
  - A distributed tree or graph that supports iteration
- Preferred way of writing simple computations:

```
A = B + alpha * C;
```

Chapel View of Compiler Transform

Initial Code:

```
A = B + alpha * C;
```

1. Promotion of scalar multiplication:

```
A = B + [c in C] alpha*c;
```

2. Promotion of scalar addition:

```
A = [(b,f) in (B,[c in C] alpha*c)] b+f;
```

3. Collapse of foralls:

```
A = [(b,c) in (B,C)] b+alpha*c;
```

4. Expansion of assignment:

```
forall (a,f) in (A,[(b,c) in (B,C)] b+alpha*c) do
  a=f;
```

5. Collapse of foralls:

```
forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

Idiom 3: Asynchronous Remote Tasks

```
on loc do begin f();
```

- Syntax

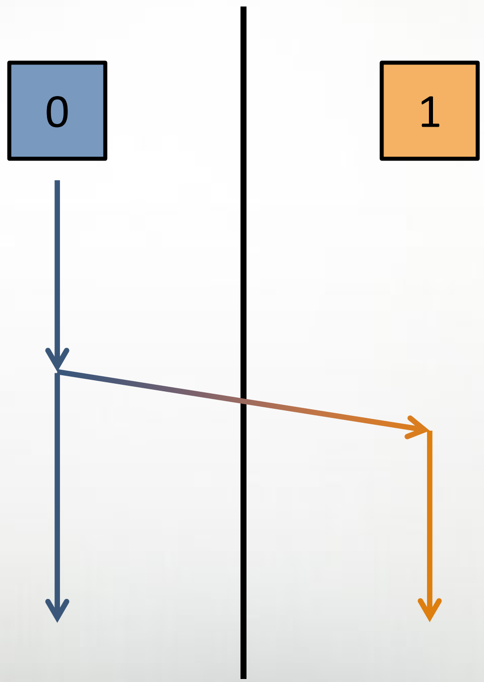
```
on expr do stmt
begin stmt
```

- Semantics
  - On-statement evaluates locale of *expr*
    Then executes *stmt* on that locale
  - Begin-statement creates a new task to execute *stmt*
    Original task continues with the next statement

Idiom 3: Asynchronous Remote Tasks

```
on loc do begin f();
```

- Picture



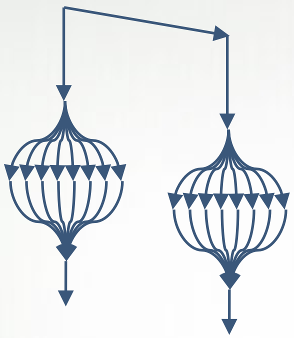
Locales vs. Tasks

- Locales
  - Abstraction of memory and processing capability
  - Architecture-dependent definition optimizes local accesses
- Tasks
  - Abstraction of computation or thread
  - Execution is *on* a locale
- Programming model support

Chapel	OpenMP	MPI	UPC	CAF	Titanium
Locales		Processes	Threads	Images	Demesnes
Tasks	Threads				

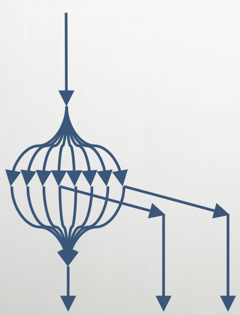
Idiom 4: Nested Parallelism

- Task parallelism of data parallelism



```
begin
  forall (a, b, c) in (A, B, C) do
    a = b + alpha * c;
forall (d, e, f) in (D, E, F) do
  d = e + beta * f;
```

- Data parallelism of task parallelism



```
forall i in D do
  if i >= 0 then
    A(i) = f(i);
  else
    on A(i) do begin A(i) = g(i);
```

Idiom 5: Remote Transactions

```
on A(i) do atomic A(i) = A(i) ^ i;
```

- Syntax

```
atomic stmt
```

- Semantics
  - Executes *stmt* with transaction semantics so that *stmt* appears to take effect atomically

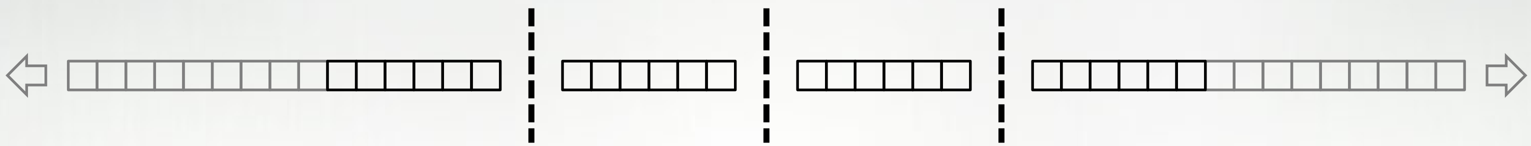
Note: atomic statements are not implemented

Outline

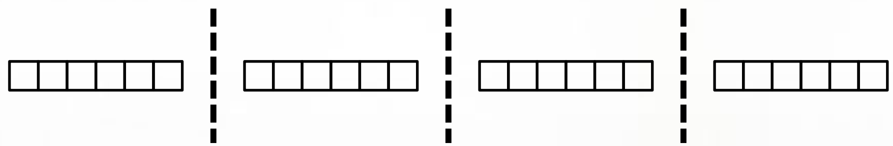
- What is Chapel
- The Five Idioms
- Performance Study
  - HPCC Global Stream
  - HPCC EP Stream

Global STREAM Triad in Chapel (Excerpts)

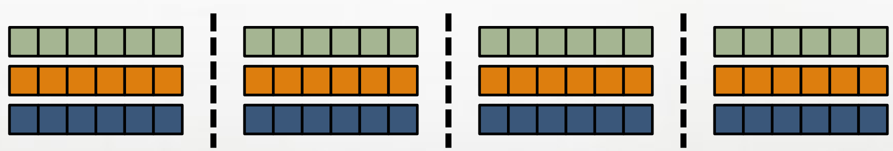
```
const BlockDist = new dmap(new Block([1..m]));
```



```
const ProblemSpace:
        domain(1,int(64)) dmapped BlockDist = [1..m];
```



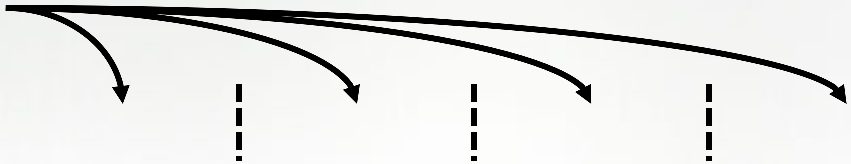
```
var A, B, C: [ProblemSpace] real;
```



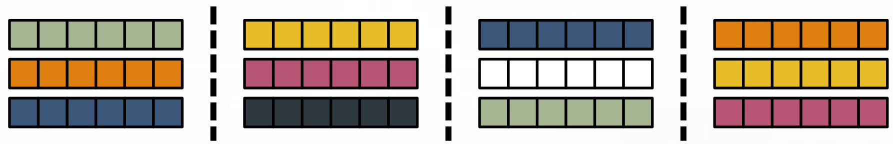
```
forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

EP STREAM Triad in Chapel (Excerpts)

```
coforall loc in Locales do on loc {
```



```
  local {
    var A, B, C: [1..m] real;
```



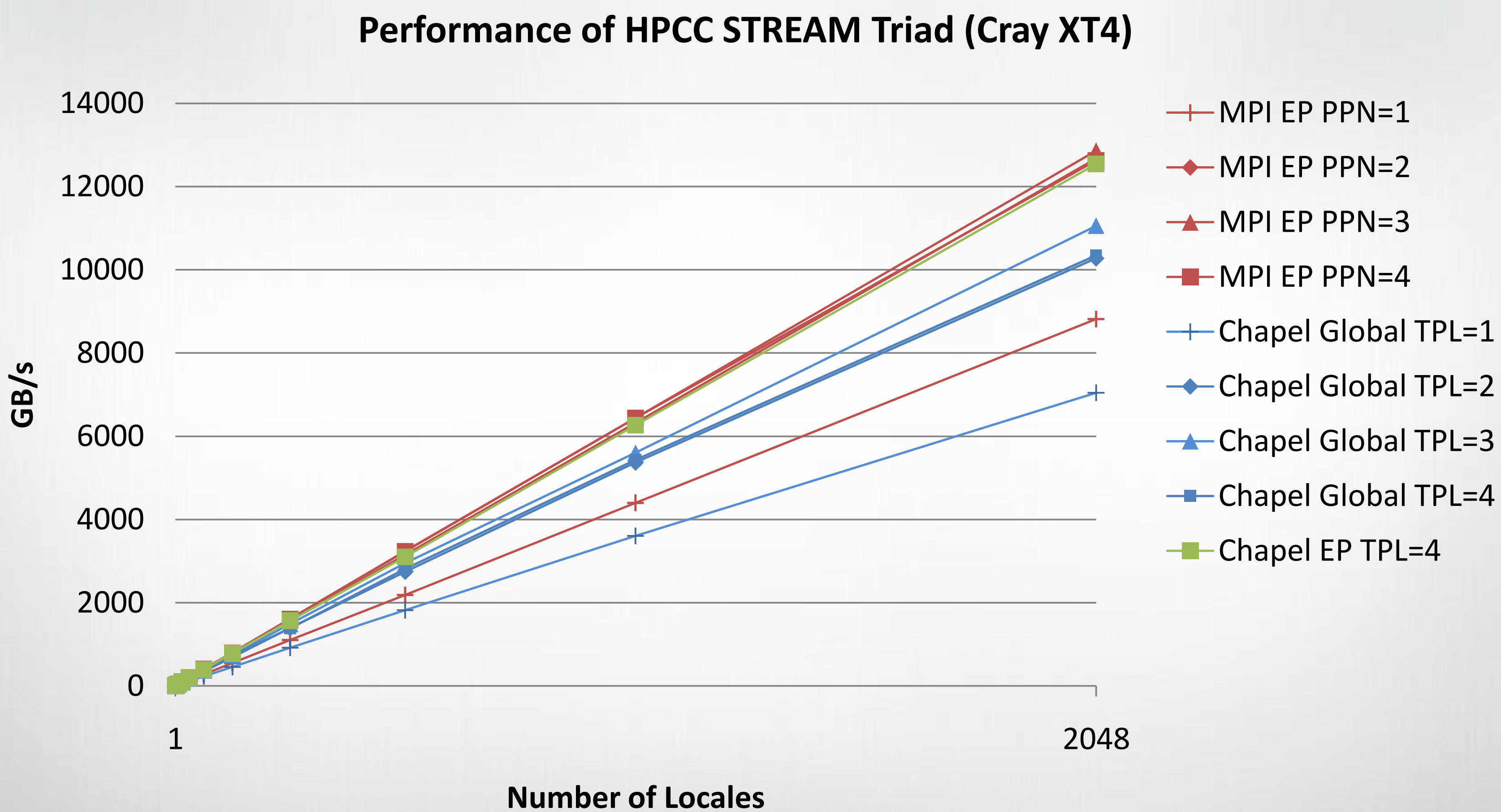
```
    forall (a,b,c) in (A,B,C) do
      a = b + alpha * c;
  }
}
```

Experimental Setup

Machine Characteristics	
Model	Cray XT4
Location	ORNL
Nodes	7832
Processor	2.1 GHz Quadcore AMD Opteron
Memory	8 GB per node

Benchmark Parameters	
STREAM Triad Memory	Least value greater than 25% of memory
Random Access Memory	Least power of two greater than 25% of memory
Random Access Updates	2^{n-10} for memory equal to 2^n

STREAM Triad Performance



Thank You

Chapel URL: http://chapel.cray.com/

Chapel Source: http://sourceforge.net/projects/chapel

Contact: chapel_info@cray.com