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://chapel.cray.com/)
    - [http://sourceforge.net/projects/chapel/](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**

  \[\text{domain-expr \texttt{dmapped} distribution-expr}\]

- **Semantics**
  - Index set of \textit{domain-expr} is partitioned via \textit{distribution-expr}
  - Partitioned across ‘locales’ of a system
  - Locale – abstraction of memory and processing capability
Idiom 1: Data Distributions

- Standard Block distribution

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

Diagram:

- **D**
- **A**
- **DD**
- **DA**
- **Locales**

0 1 2 3
Idiom 1: Data Distributions

- Standard Cyclic distribution

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

![Diagram of data distributions and locales]
• User-defined MyBanded distribution

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

![Diagram of data distributions and locales]
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
Example 1: Non-distributed arrays

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

\[
\text{forall } (a, b, c) \text{ in } (A, B, C) \text{ do} \\
a = b + \alpha \ast c;
\]

- Example 4: 2D Block-distributed arrays

![Diagram showing 2D Block-distributed arrays with Locales](image-url)
Idiom 2: Data-Parallel Loops

```chapel
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:
  ```chapel
  A = B + alpha * C;
  ```
Chapel View of Compiler Transform

Initial Code:

\[ A = B + \text{alpha} \times C; \]

1. Promotion of scalar multiplication:

\[ A = B + [c \text{ in } C] \text{alpha} \times c; \]

2. Promotion of scalar addition:

\[ A = [(b,f) \text{ in } (B,[c \text{ in } C] \text{alpha} \times c)] \text{b+f}; \]

3. Collapse of foralls:

\[ A = [(b,c) \text{ in } (B,C)] \text{b+alpha} \times c; \]

4. Expansion of assignment:

\[
\text{forall} \ (a,f) \text{ in } (A,[(b,c) \text{ in } (B,C)] \text{b+alpha} \times c) \text{ do } a=f;
\]

5. Collapse of foralls:

\[
\text{forall} \ (a,b,c) \text{ in } (A,B,C) \text{ do } a = b + \text{alpha} \times c;
\]
Idiom 3: Asynchronous Remote Tasks

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

- **Syntax**
  ```chapel
  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

```plaintext
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**

<table>
<thead>
<tr>
<th>Chapel</th>
<th>OpenMP</th>
<th>MPI</th>
<th>UPC</th>
<th>CAF</th>
<th>Titanium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locales</td>
<td></td>
<td>Processes</td>
<td>Threads</td>
<td>Images</td>
<td>Demesnes</td>
</tr>
<tr>
<td>Tasks</td>
<td>Threads</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Idiom 4: Nested Parallelism

- Task parallelism of data parallelism

```chapel
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;
end
```

- Data parallelism of task parallelism

```chapel
forall i in D do
    if i >= 0 then
        A(i) = f(i);
    else
        on A(i) do begin
            A(i) = g(i);
        end
end
```
Idiom 5: Remote Transactions

\[
\text{on } A(i) \text{ do atomic } A(i) = A(i) \text{ } ^{\land} \text{ } i;
\]

- Syntax
  
  \text{atomic } stmt

- Semantics
  
  - Executes \textit{stmt} with transaction semantics so that \textit{stmt} appears to take effect atomically

\textit{Note: atomic statements are not implemented}
Outline

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

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Cray XT4</td>
</tr>
<tr>
<td>Location</td>
<td>ORNL</td>
</tr>
<tr>
<td>Nodes</td>
<td>7832</td>
</tr>
<tr>
<td>Processor</td>
<td>2.1 GHz Quadcore AMD Opteron</td>
</tr>
<tr>
<td>Memory</td>
<td>8 GB per node</td>
</tr>
</tbody>
</table>

## Benchmark Parameters

<table>
<thead>
<tr>
<th>Benchmark Parameter</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>STREAM Triad Memory</td>
<td>Least value greater than 25% of memory</td>
</tr>
<tr>
<td>Random Access Memory</td>
<td>Least power of two greater than 25% of memory</td>
</tr>
<tr>
<td>Random Access Updates</td>
<td>$2^{n-10}$ for memory equal to $2^n$</td>
</tr>
</tbody>
</table>
Performance of HPCC STREAM Triad (Cray XT4)

- MPI EP PPN=1
- MPI EP PPN=2
- MPI EP PPN=3
- MPI EP PPN=4
- Chapel Global TPL=1
- Chapel Global TPL=2
- Chapel Global TPL=3
- Chapel Global TPL=4
- Chapel EP TPL=4
Thank You

Chapel URL:  http://chapel.cray.com/
Chapel Source:  http://sourceforge.net/projects/chapel
Contact:  chapel_info@cray.com