Programming Languages

• High-Performance Computing as Paradigm

• Lots of design choices in Chapel to discuss:
  – Task Creation (instead of Threads) with 'begin'.
  – Task Synchronicity with 'sync' and cobegin
  – Parallel loops: forall and coforall
  – Thread safety using variable 'sync'
  – reduce overcomes bottleneck
PL: Task Generation

```plaintext
var total = 0;
for i in 1..100 do total += i;

writeln('Sum is ', total, '.');

We can add a Timer to measure running time!
```
PL: Task Generation

```plaintext
var total = 0;
for i in 1..100 do total += i;

writeln('Sum is ', total, '.');

We can add a Timer to measure running time!

use Time;
var timer: Timer;
var total = 0;
timer.start();
for i in 1..100 do total += i;
timer.stop();

writeln('Sum is ', total, '.');
writeln('That took ', timer.elapsed(), ' seconds.');
```
PL: Task Generation

Now let's use another thread!

```plaintext
use Time;
var timer: Timer;
var total = 0;
var highTotal = 0;
var lowTotal = 0;
timer.start();
begin ref(highTotal) {
    for i in 51..100 do highTotal += i;
}
for i in 1..50 do lowTotal += i;
total = lowTotal + highTotal;
timer.stop();

writeln('Sum is ', total, '.');
writeln('That took ', timer.elapsed(), ' seconds.');
```

Note: ref(highTotal) at begin
PL: Task Generation

Now let's use another thread!

```plaintext
use Time;
var timer: Timer;
var total = 0;
var highTotal = 0;
var lowTotal = 0;
timer.start();
begin ref(highTotal) {
    for i in 51..100 do highTotal += i;
}
for i in 1..50 do lowTotal += i;
total = lowTotal + highTotal;
timer.stop();

writeln('Sum is ', total, '.');
writeln('That took ', timer.elapsed(), ' seconds.');
```

Result: faster, but sometimes incorrect.
PL: Synchronization

Incorrect: top thread may not finish.

Chapel provides a solution: sync

```plaintext
sync {
    begin {
        ...
    }
    begin {
        ...
    }
    ...
}
```
PL: Synchronization

Use sync:

... 

timer.start();

sync {
    begin ref(highTotal) {
        for i in 51..100 do highTotal += i;
    }
    begin ref(lowTotal) {
        for i in 1..50 do lowTotal += i;
    }
}

total = lowTotal + highTotal;

...
PL: Syntactic Sugar

Ask students: How common is this?

```plaintext
sync {
    begin {
        //single line of code
    }
    begin {
        //another single line
    }
    . . .
    begin {
        //even yet another single line
    }
}
```

So, what did language designers do?
PL: Syntactic Sugar

cobegin {
    //single line of code
    //another single line
    ...  
    //even yet another single line
}

PL: forall

forall: data-parallel loop

```plaintext
var sum = 0;
forall i in 1..100 {
    sum += i;
}
writeln("Sum is: ", sum, ".");
```
PL: forall

forall: data-parallel loop

```plaintext
var sum = 0;
forall i in 1..100 {
    sum += i;
}
writeln("Sum is: ", sum, ".");
```

Ask: Why doesn't this work?
PL: HPC Concepts

• Why doesn't it work?
  – Race conditions
  – Atomicity
  – Synchronization solutions
PL: forall

One solution: synchronized variables

```plaintext
var sum : sync int;
sum = 0;
forall i in 1..100 {
    sum += i;
}
writeln("Sum is: ", sum, ".");
```
PL: sync bottleneck and reduce

• sync causes a bottleneck:
  – Running time still technically linear.

• Reductions:
  – Divide-and-conquer solution
PL: Reductions

8  -1  2  21  1  -5  12  3
PL: Reductions

7

23

-4

15

8 -1 2 21 1 -5 12 3
PL: Reductions
PL: Reductions

```
41
/  \
30   11
/  \
7 23 /  \
/  \
8 -1 2 21 1 -5 12 3
```
PL: sync bottleneck and reduce

• sync causes a bottleneck:
  – Running time still technically linear.

• Reductions:
  – Divide-and-conquer solution
PL: sync bottleneck and reduce

• sync causes a bottleneck:
  – Running time still technically linear.

• Reductions:
  – Divide-and-conquer solution
  – Simplify with 'reduce' keyword!
PL: sync bottleneck and reduce

```plaintext
var integers : [1..100] int;
forall i in integers.domain {
    integers[i] = i;
}
var sum = + reduce integers;
```
PL: sync bottleneck and reduce

```plaintext
var integers : [1..100] int;
forall i in integers.domain {
    integers[i] = i;
}
var sum = + reduce integers;
```

One line solution?
PL: sync bottleneck and reduce

```javascript
var sum = + reduce (1..100);
```
PL: sync bottleneck and reduce

All intermediate values?

```javascript
var sum = + scan array;
```

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>-1</td>
<td>2</td>
<td>21</td>
<td>1</td>
<td>-5</td>
<td>12</td>
<td>3</td>
</tr>
</tbody>
</table>
PL: sync bottleneck and reduce

All intermediate values?

```
var sum = + scan array;
```

```
8  -1  2  21  1  -5  12  3
8  7  9  30  31  26  38  41
```
PL: sync bottleneck and reduce

Fun Uses!

```plaintext
var factorials = * scan (1..10);

var threes : [1..10] int;
forall i in threes.domain do 
  threes[i] = 3;
var powersOfThree = * scan threes;
```
Chapel Ranges

• What is a range?
• How are ranges used?
• Range operations
Chapel Ranges

• What is a range?
  – A range of values
  – Ex: var someNaturals : range = 0..50;

• How are they used?
  • Indexes for Arrays
  • Iteration space in loops

• Are there cool operations?
Chapel Ranges

• What is a range?
  – A range of values
  – Ex: var someNaturals : range = 0..50;

• How are they used?
  • Indexes for Arrays
  • Iteration space in loops

• Are there cool operations?
  Yes!
Range Operation Examples

var someNaturals: range = 0..50;
var someEvens = someNaturals by 2;
    (someEvens: 0, 2, 4, ..., 48, 50)
var someOdds = someEvens align 1;
    (someOdds: 1, 3, 5, 7, ..., 47, 49)
var fewerOdds = someOdds # 6;
    (fewerOdds: 1, 3, 5, 7, 9, 11)
Other Cool Range Things

• Can create “infinite” ranges:
  var naturals: range = 0..;

• Ranges in the “wrong order” are auto-empty:
  var nothing: range = 2..-2;

• Otherwise, negatives are just fine
Chapel Domains

• What is a domain?
• How are domains used?
• Operations on domains
• Example: Game of Life
Chapel Domains

• Domain: index set
  – Used to simplify addressing
  – Every array has a domain to hold its indices
  – Can include ranges or be sparse

• Example:
  
  ```chal
  var A: [1..10] int; //indices are 1, 2, ..., 10
  ...
  for i in A.domain {
    //do something with A[i]
  }
  ```
Chapel Domains

Array (hierarchy)

Array Domain (indices) —> Array Values
Chapel Domains

Array (hierarchy)

Domain

(Sparse)

Array Values
Chapel Domains

Array (hierarchy)

0, 2, 4, 6, …, 6000

(Range)

Domain

Array Values
Chapel Domains

Array (hierarchy)

Domain

2-D
0, 2, 4, 6, ..., 6000
1,
2,
3,
...
50

(2 Ranges)

Array Values
Chapel Domains

Array (hierarchy)

Domain

0, 2, 4, 6, ..., 6000

(Combo)

Array Values
Chapel Domains

• Domain Declaration:
  – var D: domain(2) = {0..m, 0..n};
    • D is 2-D domain with (m+1) x (n+1) entries
  – var A: [D] int;
    • A is an array of integers with D as its domain
Chapel Domains

• Domain Declaration:
  – var D: domain(2) = {0..m, 0..n};
    • D is 2-D domain with (m+1) x (n+1) entries
  – var A: [D] int;
    • A is an array of integers with D as its domain

Why is this useful?
Chapel Domains

- Changing D changes A automatically!
- $D = \{1..m, 0..n+1\}$
  - decrements height; increments width!
  - (adds zeroes)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

|   | 4 | 5 | 6 | 0 |
|---|---|---|---|
| 7 | 8 | 9 | 0 |
Domains vs. Ranges

• Despite how similar they seem so far, domains and ranges are different
  – Domains remain tied to arrays so that resizing the domain resizes the array:

```plaintext
var R : range = 1..10;
var A : [R] int;
R = 0..10; //no effect on array
A[0] = 5; //runtime error
```

```plaintext
var D : domain(1) = {1..10};
var A : [D] int;
D = 0..10; //resizes array
A[0] = 5; //ok
```

• Domains are more general; some are not sets of integers
Domain Slices (Intersection)

domain0: {0..2, 1..3}
domain1: {1..3, 3..5}
Domain Slices (Intersection)

domain0: \{0..2, 1..3\}
domain1: \{1..3, 3..5\}

domain2: \{1..2, 3..3\}
Domain Slices (Intersection)

//domain2 is the intersection of domain1 and domain0
var domain2 = domain1 [domain0];

domain0: {0..2, 1..3}
domain1: {1..3, 3..5}

domain2: {1..2, 3..3}
Domain Slices (Intersection)

//domain2 is the intersection of domain1 and domain0
var domain2 = domain1 [domain0];
PL: Projects

• Matrix Multiplication
  – Matrix-vector multiplication in class
  – Different algorithms:
    • Column-by-column
    • One entry at a time

• Collatz conjecture testing
  – Generate lots of tasks (coforall)
  – How to synchronize?
PL: Takeaways

• Lots of language features to discuss!

• Learning HPC ↔ Motivates Syntax

• Students love it!