Chapel: Heat Transfer (+ X10/Fortress)

Brad Chamberlain
Cray Inc.

CSEP 524
May 20, 2010
Heat Transfer in Pictures

\[ A : \sum_{\text{repeat until max change } < \varepsilon} \left( \sum_{\text{1.0}} \right) \div 4 \]
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
                      + A(i,j-1) + A(i,j+1)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
A[LastRow] = 1.0;
do {
    [(i,j) in D]
        Temp(i,j) = (A(i-1,j) + A(i+1,j) + A(i,j-1) + A(i,j+1)) / 4.0;
    const delta = max reduce abs(A(D) - Temp(D));
    A[D] = Temp[D];
} while (delta > epsilon);
writeln(A);
```

Declare program parameters

- `config` ⇒ can be set on executable command-line
  ```
prompt> jacobi --n=10000 --epsilon=0.0001
  ```

- `const` ⇒ can’t change values after initialization

Note that no types are given; inferred from initializer

- `n` ⇒ `integer` (current default, 32 bits)
- `epsilon` ⇒ `floating-point` (current default, 64 bits)
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);
```

**Declare domains (first class index sets)**

- `domain(2)` ⇒ 2D arithmetic domain, indices are integer 2-tuples
- `subdomain(P)` ⇒ a domain of the same type as `P` whose indices are guaranteed to be a subset of `P`'s

**Diagrams**

- `BigD`:
  - Domain of size `n+1` by `n+1`
- `D`:
  - Subdomain of size `n` by `n`
- `LastRow`:
  - Subdomain of size `n` by `1`

**exterior** ⇒ one of several built-in domain generators
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
```

### Declare arrays

- **var** ⇒ can be modified throughout its lifetime
- : **T** ⇒ declares variable to be of type **T**
- : **[D] T** ⇒ array of size **D** with elements of type **T**
  - *(no initializer)* ⇒ values initialized to default value (0.0 for reals)
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;
```

Set Explicit Boundary Condition

indexing by domain ⇒ slicing mechanism
array expressions ⇒ parallel evaluation
Heat Transfer in Chapel

Compute 5-point stencil

\[ [(i,j) \text{ in } D] \Rightarrow \text{parallel forall expression over } D\text{'s indices, binding them to new variables } i \text{ and } j \]

**Note:** since \((i,j) \in D\) and \(D \subseteq \text{BigD}\) and \(\text{Temp} : [\text{BigD}]\) 
\(\Rightarrow \text{no bounds check required for } \text{Temp}(i,j)\) with compiler analysis, same can be proven for \(A\text{'s accesses}\)

\[
\sum \left( \begin{array}{c}
\text{orange} \\
\text{yellow} \\
\text{orange}
\end{array} \right) \div 4 \\
\rightarrow \\
\begin{array}{c}
\text{blue} \\
\text{light blue}
\end{array}
\]

\[
[(i,j) \text{ in } D] \text{ Temp}(i,j) = (A(i-1,j) + A(i+1,j) + A(i,j-1) + A(i,j+1)) / 4;
\]

```chapel
const delta = max reduce abs(A[D] - Temp[D]);
A[D] = Temp[D];
} while (delta > epsilon);
writeln(A);
```
Heat Transfer in Chapel

```chapel
config const n = 6,
  epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],

Compute maximum change

op reduce ⇒ collapse aggregate expression to scalar using op

Promotion: abs() and – are scalar operators, automatically promoted to
work with array operands

do {
  [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
  + A(i,j-1) + A(i,j+1)) / 4;

  const delta = max reduce abs(A[D] - Temp[D]);
  A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
config const n = 6, 
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
A[LastRow] = 1.0; 
do 
    [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j) 
        + A(i,j-1) + A(i,j+1)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);
writeln(A);

Copy data back & Repeat until done

uses slicing and whole array assignment
standard do...while loop construct
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1],
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [i, j] in D
        Temp(i, j) = (A(i-1, j) + A(i+1, j) + A(i, j-1) + A(i, j+1)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```

Write array to console

If written to a file, parallel I/O would be used
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;
```

With this change, same code runs in a distributed manner

Domain distribution maps indices to *locales*

⇒ decomposition of arrays & default location of iterations over locales

Subdomains inherit parent domain's distribution
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j)
        + A(i,j-1) + A(i,j+1)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
Heat Transfer in Chapel (Variations)
Heat Transfer in Chapel (double buffered version)

```chapel
config const n = 6,
            epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A : [1..2] [BigD] real;
A[..][LastRow] = 1.0;

var src = 1, dst = 2;

do {
    [(i,j) in D] A(dst)(i,j) = (A(src)(i-1,j) + A(src)(i+1,j)
                        + A(src)(i,j-1) + A(src)(i,j+1)) / 4;

    const delta = max reduce abs(A[src] - A[dst]);
    src <=> dst;
} while (delta > epsilon);

writeln(A);
```
Heat Transfer in Chapel (ZPL style)

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

const north = (-1,0), south = (1,0), east = (0,1), west = (0,-1);

var A, Temp : [BigD] real;
A[LastRow] = 1.0;

do {
    [ind in D] Temp(ind) = (A(ind + north) + A(ind + south)
        + A(ind + east) + A(ind + west)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
Heat Transfer in Chapel (array of offsets version)

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

param offset : [1..4] (int, int) = ((-1,0), (1,0), (0,1), (0,-1));

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [ind in D] Temp(ind) = (+ reduce [off in offset] A(ind + off))
    / offset.numElements;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
Heat Transfer in Chapel (sparse offsets version)

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

param stencilSpace: domain(2) = [-1..1, -1..1],
    offSet: sparse subdomain(stencilSpace)
        = ((-1,0), (1,0), (0,1), (0,-1));

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [ind in D] Temp(ind) = (+ reduce [off in offSet] A(ind + off))
        / offSet.numIndices;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
The Other HPCS Languages
X10 in a Nutshell

- Heavily influenced by Java, Scala
  - emphasis on type safety, OOP design, small core language
  - also ZPL: support for global-view domains and arrays

- Similar concepts to what you’ve heard about today in Chapel
  - yet a fairly different syntax and design aesthetic

- Main differences from Chapel

- For more information:
  - [http://x10-lang.org/](http://x10-lang.org/)
  - [http://sf.net/projects/x10](http://sf.net/projects/x10)
  - [http://dist.codehaus.org/](http://dist.codehaus.org/)
X10: Similarities to Chapel

- **PGAS memory model**
  - plus, language concepts for referring to realms of locality

- **more dynamic (“post-SPMD”) execution model**
  - one logical task executes main()
  - any task can create additional tasks--local or remote

- **global-view data structures**
  - ability to declare and access distributed arrays holistically rather than piecemeal

- **many similar concepts, often with different names/semantics**
  - tasks vs. tasks
  - places vs. locales
  - ‘at’ vs. ‘on’
  - ‘ateach’ vs ‘coforall’ + ‘on’
  - ‘async’ vs. ‘begin’
  - ‘finish’ vs. ‘sync’
X10: Differences from Chapel

- X10:
  - takes a purer object-oriented approach
    - for example, arrays have reference rather than value semantics
      
      \[ A = B; \quad \text{// alias or copy if } A \text{ and } B \text{ are arrays?} \]
    - based on Java/Scala rather than *ab initio*
      - reflects IBM’s customer base relative to Cray’s
  - a bit more minimalist and purer
    - e.g., less likely to add abstractions to the language if expressible using objects
  - semantics distinguish between local and remote more strongly
    - e.g., communication is more visible in the code
    - e.g., some operations are not legal on remote objects
    - reflect differing choices on orthogonality vs. performance/safety
  - has a stronger story for exceptions
class HeatTransfer_v2 {
  const BigD = Dist.makeBlock([0..n+1, 0..n+1], 0);
  const D = BigD | ([1..n, 1..n] as Region);
  const LR = [0..0, 1..n] as Region;
  const A = DistArray.make[double](BigD,(p:Point)=>{ LR.contains(p) ? 1 : 0 });
  const Temp = DistArray.make[double](BigD);
  static def stencil_1((x,y):Point(2)) {
    return ((at(A.dist(x-1,y)) A(x-1,y)) +
    (at(A.dist(x+1,y)) A(x+1,y)) +
    A(x,y-1) + A(x,y+1)) / 4;
  }
  def run() {
    val D_Base = Dist.makeUnique(D.places());
    var delta:double = 1.0;
    do {
      finish ateach (z in D_Base)
      for (p:Point(2) in D | here)
        Temp(p) = stencil_1(p);
        delta = A.lift(Temp, D.region, (x:double,y:double)
        =>Math.abs(x-y)).reduce(Math.max.(Double,Double), 0);
      finish ateach (p in D) A(p) = Temp(p);
    } while (delta > epsilon);
  }
}
Heat Transfer in X10
Heat Transfer in Chapel

```chapel
config const n = 6,
    epsilon = 1.0e-5;

const BigD: domain(2) = [0..n+1, 0..n+1] dmapped Block,
    D: subdomain(BigD) = [1..n, 1..n],
    LastRow: subdomain(BigD) = D.exterior(1,0);

var A, Temp : [BigD] real;

A[LastRow] = 1.0;

do {
    [(i,j) in D] Temp(i,j) = (A(i-1,j) + A(i+1,j) + A(i,j-1) + A(i,j+1)) / 4;

    const delta = max reduce abs(A[D] - Temp[D]);
    A[D] = Temp[D];
} while (delta > epsilon);

writeln(A);
```
Fortress in a Nutshell

- The most blue-sky, clean-slate of the HPCS languages

**Goal:** define language semantics in libraries, not compiler:
  - data structures and types (including scalars types?)
  - operators, typecasts
  - operator precedence
  - in short, as much as possible to support future changes, languages

- Other themes:
  - implicitly parallel -- most things are parallel by default
  - supports mathematical notation, symbols, operators
  - functional semantics
  - hierarchical representation of target architecture’s structure
  - units of measurement in the type system (meters, seconds, miles, …)

- For more information:
  - [http://projectfortress.sun.com/Projects/Community/](http://projectfortress.sun.com/Projects/Community/)