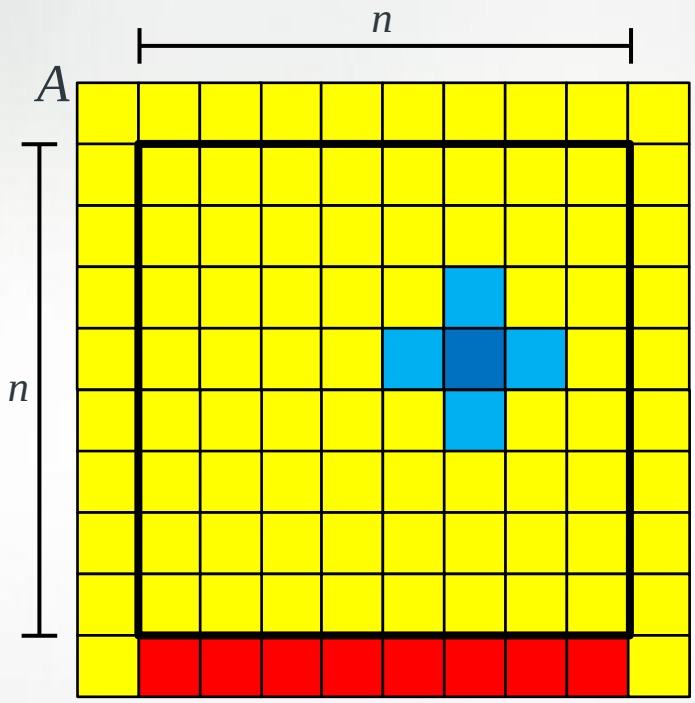
CRAY
THE SUPERCOMPUTER COMPANY

Heat Transfer in Chapel

Steve Deitz

Cray Inc.

Heat Transfer in Pictures



repeat until max change < ε

$$\sum \left(\right) \div 4 \Rightarrow$$

Heat Transfer in 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

```
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[LastR

do {
  [(i,j

  const
  A[D]
} while

writeln
```

Declare program parameters

const ⇒ can't change values after initialization

config ⇒ can be set on executable command-line

```
prompt> a.out --n=10000 --epsilon=0.0001
```

note that no types are given; inferred from initializer

n ⇒ **integer** (current default, 32 bits)

epsilon ⇒ **floating-point** (current default, 64 bits)

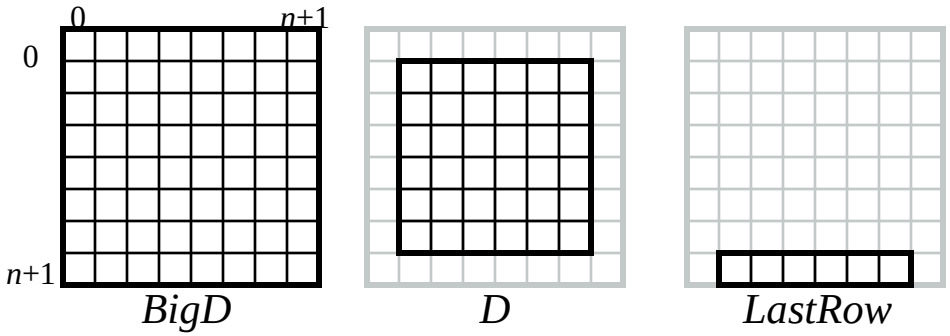
Heat Transfer in Chapel

```
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



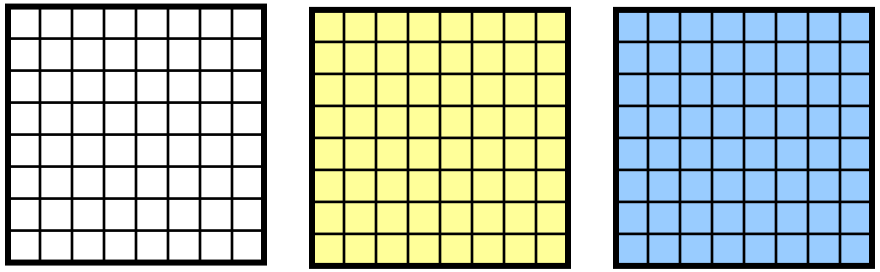
exterior ⇒ one of several built-in domain generators

Heat Transfer in 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 with indexes from *D* and elements of type *T*
(no initializer) ⇒ values initialized to default value (0.0 for reals)

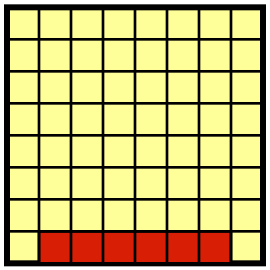


Heat Transfer in Chapel

```
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
```

Set Explicit Boundary Condition

indexing by domain ⇒ slicing mechanism
array expressions ⇒ parallel evaluation



Heat Transfer in Chapel

Compute 5-point stencil

[(*i,j*) in *D*] ⇒ parallel forall expression over *D*'s indices, binding them to new variables *i* and *j*

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

```
  [(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

```
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);
```

Heat Transfer in 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,

A[Last

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

```
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,
                                          ) / 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

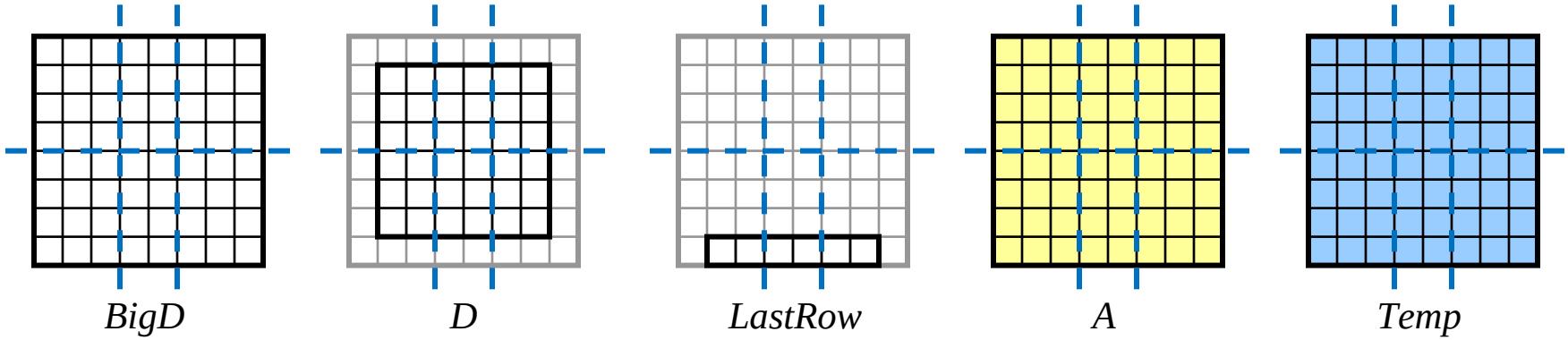
```
config const n = 6, epsilon = 1.0e-5;

const BigD: domain(2) dmapped Block([1..n,1..n]) = [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;
```

With this change, same code runs in a distributed manner

Block domain map partitions indices across *locales*

⇒ decomposition of arrays & default location of iterations over locales

Subdomains inherit parent domain's distribution



Heat Transfer in Chapel

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

const BigD: domain(2) dmapped Block([1..n,1..n]) = [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);
```

CRAY
THE SUPERCOMPUTER COMPANY

Variations

Heat Transfer in Chapel (double buffering)

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

const BigD: domain(2) dmapped Block([1..n,1..n]) = [0..n+1,0..n+1],
          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][D] - A[dst][D]);
  src <=> dst;
} while (delta > epsilon);

writeln(A[src]);
```

Heat Transfer in Chapel (named directions)

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

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

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

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

const 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 (domain of offsets)

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

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

const 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);
```

http://chapel.cray.com/ ◆ http://sourceforge.net/projects/chapel ◆ chapel_info@cray.com