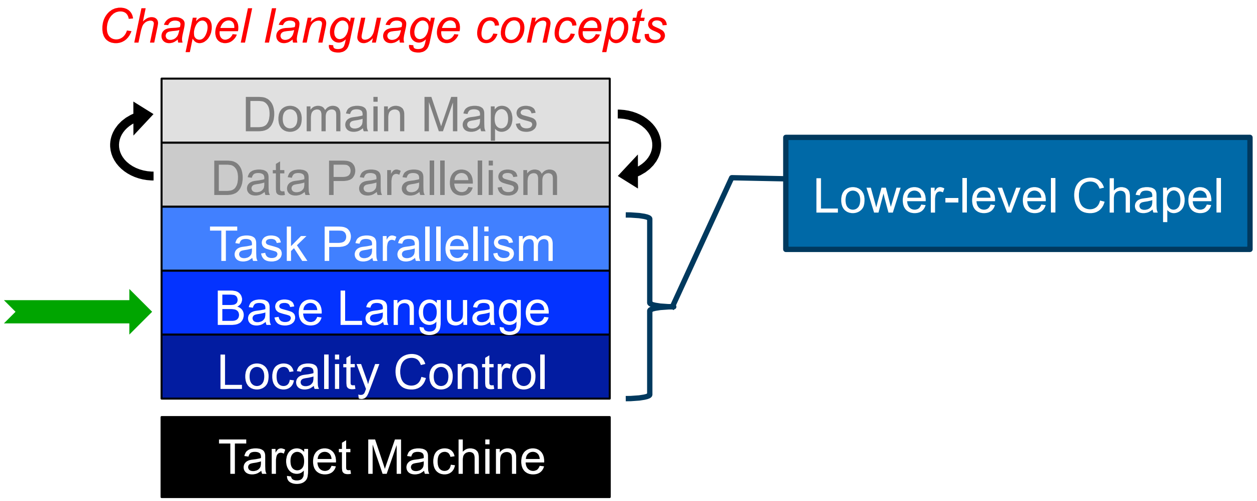
Chapel Base Language, By Example

Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

Lower-Level Features



Chapel language concepts

Domain Maps

Data Parallelism

Task Parallelism

Base Language

Locality Control

Target Machine

Lower-level Chapel

"Hello World" in Chapel: Two Versions

- Fast prototyping

```
writeln(“Hello, world!”);
```

- “Production-grade”

```
module Hello {


  proc main() {
    writeln(“Hello, world!”);
  }
}
```

"Hello World" in Chapel: Two Versions

- Fast prototyping

```
writeln(“Hello, world!”);
```

- “Production-grade” (configurable)

```
module Hello {
  config const audience = “world”;

  proc main() {
    writeln(“Hello, ”, audience, “!”);
  }
}
```

Static Type Inference

```
const pi = 3.14,           // pi is a real
      coord = 1.2 + 3.4i,  // coord is a complex…
      coord2 = pi*coord,   // …as is coord2
      name = “brad”,       // name is a string
      verbose = false;     // verbose is boolean

proc addem(x, y) {         // addem() has generic arguments
  return x + y;            //   and an inferred return type
}

var sum = addem(1, pi),              // sum is a real
    fullname = addem(name, “ford”);  // fullname is a string

writeln((sum, fullname));
```

```
(4.14, bradford)
```

Variables, Constants, and Parameters

- **Basic syntax**

```
declaration:
  var   identifier [: type] [= init-expr];
  const identifier [: type] [= init-expr];
  param identifier [: type] [= init-expr];
```

- **Meaning**
  - `var`/`const`: execution-time variable/constant
  - `param`: compile-time constant
  - No *init-expr* ⇒ initial value is the type's default
  - No *type* ⇒ type is taken from *init-expr*
- **Examples**

```
const pi: real = 3.14159;
var count: int;           // initialized to 0
param debug = true;       // inferred to be bool
```

Configs

```
param intSize = 32;
type elementType = real(32);
const epsilon = 0.01:elementType;
var start = 1:int(intSize);
```

Configs

```
config param intSize = 32;
config type elementType = real(32);
config const epsilon = 0.01:elementType;
config var start = 1:int(intSize);
```

```
% chpl myProgram.chpl -sintSize=64 -selementType=real
% a.out --start=2 --epsilon=0.00001
```

"Hello World" in Chapel: Two Versions

- Fast prototyping

```
writeln(“Hello, world!”);
```

- “Production-grade” (configurable)

```
module Hello {
  config const audience = “world”;

  proc main() {
    writeln(“Hello, ”, audience, “!”);
  }
}
```

n-body in Chapel (for n == 5)

(a sample serial computation)

n-body in Chapel (where n == 5)

- A serial computation
- From the Computer Language Benchmarks Game
- Computes the influence of 5 bodies on one another
  - The Sun, Jupiter, Saturn, Uranus, Neptune
- Executes for a user-specifiable number of timesteps

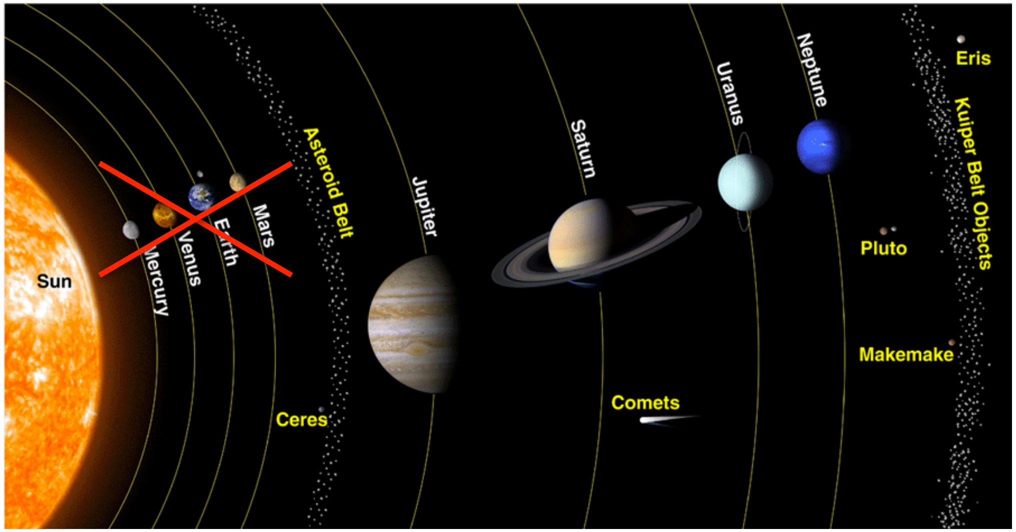


Image source: http://spaceplace.nasa.gov/review/ice-dwarf/solar-system-lrg.png

5-body in Chapel: Declarations

```
config const numsteps = 10000;

const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

record body {
  var pos: 3*real;
  var v:   3*real;
  var mass: real;
}


…
```

Records and Classes

- **Chapel's struct/object types**
  - Contain variable definitions (fields)
  - Contain procedure & iterator definitions (methods)
  - Records: value-based (*e.g.*, assignment copies fields)
  - Classes: reference-based (*e.g.*, assignment aliases object)
  - Record : Class :: C++ struct : Java class

- **Example**

```
record circle {
  var radius: real;
  proc area() {
    return pi*radius**2;
  }
}
```

```
var c1, c2: circle;
c1 = new c1(radius=1.0);
c2 = c1;           // copies c1
c1.radius = 5.0;
writeln(c2.radius);  // 1.0
// records deleted by compiler
```

Records and Classes

- **Chapel’s struct/object types**
  - Contain variable definitions (fields)
  - Contain procedure & iterator definitions (methods)
  - Records: value-based (*e.g.*, assignment copies fields)
  - Classes: reference-based (*e.g.*, assignment aliases object)
  - Record : Class :: C++ struct : Java class

- **Example**

```
class circle {
  var radius: real;
  proc area() {
    return pi*radius**2;
  }
}
```

```
var c1, c2: circle;
c1 = new c1(radius=1.0);
c2 = c1;  // aliases c1’s circle
c1.radius = 5.0;
writeln(c2.radius);  // 5.0
delete c1; // users delete classes
```

Tuples

- ## Use
  - support lightweight grouping of values
    - e.g., passing/returning procedure arguments
    - multidimensional array indices
    - short vectors

- ## Examples

```
var coord: (int, int, int) = (1, 2, 3);
var coordCopy: 3*int = coord;
var (i1, i2, i3) = coord;
var triple: (int, string, real) = (7, “eight”, 9.0);
```

5-body in Chapel: Declarations

```
config const numsteps = 10000;

const pi = 3.141592653589793,
      solarMass = 4 * pi**2,
      daysPerYear = 365.24;

record body {
  var pos: 3*real;
  var v: 3*real;
  var mass: real;
}


…
```

5-body in Chapel: the Bodies

```
var bodies =
      [/* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.84143144246472090e+00,
                       -1.16032004402742839e+00,
                       -1.03622044471123109e-01),
                  v = ( 1.66007664274403694e-03 * daysPerYear,
                        7.69901118419740425e-03 * daysPerYear,
                       -6.90460016972063023e-05 * daysPerYear),
               mass =   9.54791938424326609e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
     ]
```

Array Types

- ## Syntax

```
array-type:
  [ domain-expr ] elt-type
array-value:
  [elt1, elt2, elt3, … eltn]
```

- ## Meaning:
  - array-type: stores an element of *elt-type* for each index
  - array-value: represent the array with these values

- ## Examples

```
var A: [1..3] int = [5, 3, 9], // 3-element array of ints
    B: [1..3, 1..5] real,      // 2D array of reals
    C: [1..3][1..5] real;      // array of arrays of reals
```

Much more on arrays in data parallelism section later…

5-body in Chapel: the Bodies

```
var bodies =
      [/* sun */
       new body(mass = solarMass),

       /* jupiter */
       new body(pos = ( 4.84143144246472090e+00,
                       -1.16032004402742839e+00,
                       -1.03622044471123109e-01),
                  v = ( 1.66007664274403694e-03 * daysPerYear,
                        7.69901118419740425e-03 * daysPerYear,
                       -6.90460016972063023e-05 * daysPerYear),
               mass =   9.54791938424326609e-04 * solarMass),

       /* saturn */
       new body(…),

       /* uranus */
       new body(…),

       /* neptune */
       new body(…)
     ]
```

5-body in Chapel: main()

```
…

proc main() {
  initSun();

  writef("%.9r\n", energy());
  for 1..numsteps do
    advance(0.01);
  writef("%.9r\n", energy());
}

…
```

Ranges, by example

```
const r = 1..10;

printVals(r);
printVals(r # 3);
printVals(r by 2);
printVals(r by -2);
printVals(r by 2 # 3);
printVals(r # 3 by 2);
printVals(0.. #n);

proc printVals(r) {
  for i in r do
    write(r, “ ”);
  writeln();
}
```

```
1 2 3 4 5 6 7 8 9 10
1 2 3
1 3 5 7 9
10 8 6 4 2
1 3 5
1 3
0 1 2 3 4 … n-1
```

Range Values

- **Syntax**

```
range-expr:
  [low] .. [high]
```

- **Semantics**
  - Regular sequence of integers
    low <= high: *low*, *low*+1, *low*+2, …, *high*
    low > *high*: degenerate (an empty range)
    low or *high* unspecified: unbounded in that direction
- **Examples**

```
1..6                // 1, 2, 3, 4, 5, 6
6..1                // empty
3..                 // 3, 4, 5, 6, 7, …
```

For Loops

- **Syntax:**

```
for-loop:
  for [index-expr in] iteratable-expr { stmt-list }
```

- **Meaning:**
  - Executes loop body serially, once per loop iteration
  - Declares new variables for identifiers in *index-expr*
    - type and const-ness determined by *iteratable-expr*
    - *iteratable-expr* could be a range, array, or iterator

- **Examples**

```
var A: [1..3] string = [“DO”, “RE”, “MI”];

for i in 1..3 { write(A[i]); }          // DO RE MI
for a in A { a += “LA”; } write(A);     // DOLA RELA MILA
```

5-body in Chapel: main()

```
…

proc main() {
  initSun();

  writef("%.9r\n", energy());
  for 1..numsteps do
    advance(0.01);
  writef("%.9r\n", energy());
}

…
```

5-body in Chapel: advance()

```
 advance(0.01);
…
proc advance(dt) {
  for i in 1..numbodies {
    for j in i+1..numbodies {
      const dpos = bodies[i].pos - bodies[j].pos,
             mag = dt / sqrt(sumOfSquares(dpos))**3;

      bodies[i].v -= dpos * bodies[j].mass * mag;
      bodies[j].v += dpos * bodies[i].mass * mag;
    }
  }

  for b in bodies do
    b.pos += dt * b.v;
}
```

Procedures, by example

- Example to compute the area of a circle

```
proc area(radius: real): real {
  return 3.14 * radius**2;
}

writeln(area(2.0));   // 12.56
```

```
proc area(radius) {
  return 3.14 * radius**2;
}
```

Argument and return types can be omitted

- Example of argument default values, naming

```
proc writeCoord(x: real = 0.0, y: real = 0.0) {
  writeln((x,y));
}

writeCoord(2.0);          // (2.0, 0.0)
writeCoord(y=2.0);        // (0.0, 2.0)
writeCoord(y=2.0, 3.0);  // (3.0, 2.0)
```

5-body in Chapel: advance()

```
 advance(0.01);
…
proc advance(dt) {
  for i in 1..numbodies {
    for j in i+1..numbodies {
      const dpos = bodies[i].pos - bodies[j].pos,
             mag = dt / sqrt(sumOfSquares(dpos))**3;

      bodies[i].v -= dpos * bodies[j].mass * mag;
      bodies[j].v += dpos * bodies[i].mass * mag;
    }
  }

  for b in bodies do
    b.pos += dt * b.v;
}
```

5-body in Chapel: Using Iterators

```
iter triangle(n) {
  for i in 1..n do
    for j in i+1..n do
      yield (i,j);
}

proc advance(dt) {
  for (i,j) in triangle(numbodies) {
    const dpos = bodies[i].pos - bodies[j].pos,
          mag = dt / sqrt(sumOfSquares(dpos))**3;

    …
  }
…
}
…
```

Iterators

```
iter fibonacci(n) {
  var current = 0,
      next = 1;
  for 1..n {
    yield current;
    current += next;
    current <=> next;
  }
}
```

```
for f in fibonacci(7) do
  writeln(f);
```

```
0
1
1
2
3
5
8
```

```
iter tiledRMO(D, tilesize) {
  const tile = {0..#tilesize,
                0..#tilesize};
  for base in D by tilesize do
    for ij in D[tile + base] do
      yield ij;
}
```

```
for ij in tiledRMO({1..m, 1..n}, 2) do
  write(ij);
```

```
(1,1)(1,2)(2,1)(2,2)
(1,3)(1,4)(2,3)(2,4)
(1,5)(1,6)(2,5)(2,6)
…
(3,1)(3,2)(4,1)(4,2)
```

Zippered Iteration

```
for (i,f) in zip(0..#n, fibonacci(n)) do
  writeln(“fib #”, i, “ is ”, f);
```

```
fib #0 is 0
fib #1 is 1
fib #2 is 1
fib #3 is 2
fib #4 is 3
fib #5 is 5
fib #6 is 8
…
```

5-body in Chapel: Using Iterators

```
iter triangle(n) {
  for i in 1..n do
    for j in i+1..n do
      yield (i,j);
}

proc advance(dt) {
  for (i,j) in triangle(numbodies) {
    const dpos = bodies[i].pos - bodies[j].pos,
           mag = dt / sqrt(sumOfSquares(dpos))**3;

    …
  }
…
}
…
```

Other Base Language Features

- **rank-independent programming features**
- **interoperability features**
- **compile-time features for meta-programming**
  - e.g., compile-time functions to compute types, parameters
- **other OOP features**
- **argument intents, default values, match-by-name**
- **overloading, where clauses**
- **modules (for namespace management)**
- **…**

Questions about the Base Language?

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, URIKA, and YARCDATA. The following are trademarks of Cray Inc.: ACE, APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.

CRAY CHAPEL

CRAY

THE SUPERCOMPUTER COMPANY

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