Productive Programming in Chapel: A Computation-Driven Introduction

Base Language with n-body

Michael Ferguson and Lydia Duncan
Cray Inc,
SC15 November 15th, 2015
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.
Outline

✓ Motivation
✓ Chapel Background and Themes

➢ Learning the Base Language with n-body
● Short Introduction to Task Parallelism
● Hands-On 1: Hello World
● Short Introduction to Locality
● Data Parallelism with Jacobi
● Hands-On 2: Mandelbrot
● Project Status, Next Steps
Learning Chapel with n-body
n-body in Chapel (where n == 5)

- A serial computation
- From the Computer Language Benchmarks Game
  - Chapel implementation in release under examples/benchmarks/shootout/nbody.chpl
- 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](http://spaceplace.nasa.gov/review/ice-dwarf/solar-system-lrg.png)
const pi = 3.141592653589793,  
solarMass = 4 * pi**2,  
daysPerYear = 365.24;

config const numsteps = 10000;

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

...
5-body in Chapel: Declarations

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

config const numsteps = 10000;

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

Variable declarations

Configuration Variable

Record declaration

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

config const numsteps = 10000;

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

...
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**

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

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

config const numsteps = 10000;

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

...
5-body in Chapel: Declarations

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

config const numsteps = 10000;

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

...
Conigs

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

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

config const numsteps = 10000;

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

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

var c1, c2: circle;
c1 = new circle(radius=1.0);
c2 = c1; // copies c1
c1.radius = 5.0;
writeln(c2.radius); // 1.0
// records deleted by compiler
```
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 circle(radius=1.0);
c2 = c1; // aliases c1’s circle
c1.radius = 5.0;
writeln(c2.radius); // 5.0
delete c1; // users delete classes
const pi = 3.141592653589793, 
solarMass = 4 * pi**2, 
daysPerYear = 365.24;

config const numsteps = 10000;

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

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

● **Examples**

```plaintext
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**

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

config const numsteps = 10000;

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

- **Variable declarations**
- **Configuration**
- **Variable**
- **Record declaration**
- **Tuple type**
5-body in Chapel: the Bodies

```chapel
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: the Bodies

```chapel
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(...)
]
```

Creating a Record

Tuples

Array
5-body in Chapel: the Bodies

```chapel
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: the Bodies

```chapel
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,
                       7.6990118419740425e-03 * daysPerYear,
                       -6.90460016972063023e-05 * daysPerYear),
                  mass =  9.54791938424326609e-04 * solarMass),

  /* saturn */
  new body(...),

  /* uranus */
  new body(...),

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

**Array**

**Creating a Record**

**Tuples**
5-body in Chapel: the Bodies

```chapel
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(...) ]
```

Creating a Record

Tuples

Array
Array Types

- **Syntax**

  ```
  array-type:
  [ domain-exp ] 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, // A stores 0, 0, 0
  B = [5, 3, 9],    // B stores 5, 3, 9
  C: [1..m, 1..n] real, // 2D m by n array of reals
  D: [1..m][1..n] real; // array of arrays of reals
  ```

  *Much more on arrays in data parallelism section later…*
5-body in Chapel: the Bodies

```chapel
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(...) ]
```

Creating a Record

Tuples

Array
... proc main() {
    initSun();

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

...
proc main() {
    initSun();
    writeln("%.9r\n", energy());
    for 1..numsteps do
        advance(0.01);
        writeln("%.9r\n", energy());
}
5-body in Chapel: main()

```chapel
proc main() {
  initSun();
  writef("%.9r\n", energy());
  for 1..numsteps do
    advance(0.01);
    writef("%.9r\n", energy());
}
```

Procedure Definition

Procedure Call

Formatted I/O

*not covered here

Looping over a Range
proc main() {
    initSun();

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

● **Syntax**

\[
\text{range-} \text{expr:} \\
[\text{low}] .. [\text{high}]
\]

● **Semantics**

● Regular sequence of integers
  
  \( \text{low} \leq \text{high}: \text{low}, \text{low}+1, \text{low}+2, \ldots, \text{high} \)
  
  \( \text{low} > \text{high} \): degenerate (an empty range)
  
  \( \text{low} \) or \( \text{high} \) unspecified: unbounded in that direction

● **Examples**

\[
\begin{align*}
1..6 & \quad // \ 1, 2, 3, 4, 5, 6 \\
6..1 & \quad // \ \text{empty} \\
3.. & \quad // \ 3, 4, 5, 6, 7, \ldots
\end{align*}
\]
Range Types and Algebra

**const** \( r = 1..10; \)

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

**proc** `printVals(r) {`
  **for** `i in r do`
  `write(i, " ");`
  `writeln();`
}
5-body in Chapel: main()

```chapel
proc main() {
    initSun();
    writeln("%.9r\n", energy());
    for 1..numsteps do
        advance(0.01);
        writeln("%.9r\n", energy());
}
```

*Copyright 2015 Cray Inc.*
For Loops

● Syntax:

```chapel
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

```chapel
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()

```chapel
... 
proc main() {
    initSun();
    printf("%.9r\n", energy());
    for 1..numsteps do
        advance(0.01);
        printf("%.9r\n", energy());
    }
... 
... 
```

- Function Declaration
- Function Call
- Formatted I/O (*not covered here*)
- Looping over a Range
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: advance()

```chapel
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;
}
```

\[
m_1 a_1 = \frac{G m_1 m_2}{r_{12}^3} (r_2 - r_1) \quad \text{Sun-Earth}
\]

\[
m_2 a_2 = \frac{G m_1 m_2}{r_{21}^3} (r_1 - r_2) \quad \text{Earth-Sun}
\]
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

```chapel
proc area(radius: real): real {
    return 3.14 * radius**2;
}
writeln(area(2.0)); // 12.56
```

- Example of argument default values, naming

```chapel
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() using references

```chapel
proc advance(dt) {
    for i in 1..numbodies {
        for j in i+1..numbodies {
            ref bi = bodies[i],
                bj = bodies[j];

            const dpos = bi.pos - bj.pos,
                mag = dt / sqrt(sumOfSquares(dpos))**3;

            bi.v -= dpos * bj.mass * mag;
            bj.v += dpos * bi.mass * mag;
        }
    }

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

Reference declarations
Reference Declarations

- **Syntax:**

  ```chapel
defaultdecl:
default ident = expr;
```

- **Meaning:**
  - Causes ‘ident’ to refer to variable specified by ‘expr’
  - Subsequent reads/writes of ‘ident’ refer to that variable
  - Not a pointer: no way to reference something else with ‘ident’
  - Similar to a C++ reference

- **Examples**

  ```chapel
var A: [1..3] string = [" DO", " RE", " MI"];
default a2 = A[2];
a2 = " YO";
for i in 1..3 { write(A(i)); }  // DO YO MI
```
5-body in Chapel: advance() using references

```chapel
proc advance(dt) {
    for i in 1..numbodies {
        for j in i+1..numbodies {
            ref bi = bodies[i], bj = bodies[j];

            const dpos = bi.pos - bj.pos,
                        mag = dt / sqrt(sumOfSquares(dpos)) ** 3;

            bi.v -= dpos * bj.mass * mag;
            bj.v += dpos * bi.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

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

```plaintext
for f in fibonacci(7) do writeln(f);
0
1
1
2
3
5
8
```

```plaintext
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;
}
```

```plaintext
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)
```
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
...
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
- argument intents
- 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, and URIKA. 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.