Chapel: A Versatile Tool for Teaching Undergraduates Parallel Programming

David P. Bunde, Knox College

Kyle Burke, Colby College

Acknowledgements

- Material drawn from tutorials created with contributions from Johnathan Ebbers, Maxwell Galloway-Carson, Michael Graf, Ernest Heyder, Sung Joo Lee, Andrei Papancea, and Casey Samoore
- Incorporates suggestions from Michael Ferguson
- Work partially supported by NSF awards DUE-1044299 and CCF-0915805. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation

Topics

- Introduction to Chapel
- Chapel in Programming Languages
- Hands-on time
- Chapel in Analysis of Algorithms
- Chapel in Parallel Programming
- Hands-on time
- Final Discussion

Basic Facts about Chapel

- Parallel programming language developed with programmer productivity in mind
- Originally Cray's project under DARPA's High Productivity Computing Systems program
- Suitable for shared- or distributed memory systems
- Installs easily on Linux and Mac OS; use Cygwin to install on Windows

Why Chapel?

- Flexible syntax; only need to teach features that you need
- Provides high-level operations
- Designed with parallelism in mind

Flexible Syntax

- Supports scripting-like programs:

  writeln(“Hello World!”);

- Also provides objects and modules

Provides High-level Operations

- Reductions

  Ex: x = + reduce A //sets x to sum of elements of A

  Also valid for other operators (min, max, *, ...)

- Scans

  Like a reduction, but computes value for each prefix

  A = [1, 3, 2, 5];

  B = + scan A; //sets B to [1, 1+3=4, 4+2=6, 6+5=11]

Provides High-level Operations (2)

- Function promotion:

  B = f(A); //applies f elementwise for any function f

- Includes built-in operators:

  C = A + 1;

  D = A + B;

  E = A * B;

  …

Designed with Parallelism in Mind

- Operations on previous slides parallelized automatically
- Create asynchronous task w/ single keyword
- Built-in synchronization for tasks and variables

Your Presenters are...

- Enthusiastic Chapel users
- Interested in high-level parallel programming
- Educators who use Chapel with students

- **NOT connected to Chapel development team**

Chapel Resources

- Materials for this workshop

  http://faculty.knox.edu/dbunde/teaching/chapel/SIGCSE14/

- Our tutorials

  http://faculty.knox.edu/dbunde/teaching/chapel/

  http://cs.colby.edu/kgburke/?resource=chapelTutorial

- Chapel website (tutorials, papers, language specification)

  http://chapel.cray.com

- Mailing lists (on SourceForge)

Practice Systems

- We have practice accounts set up for use during the workshop
- Get handout from one of the instructors
- Will keep accounts available for a couple of weeks

“Hello World” in Chapel

- Create file hello.chpl containing

  writeln(“Hello World!”);

- Compile with

  chpl –o hello hello.chpl

- Run with

  ./hello

Variables and Constants

- Variable declaration format:

  [config] var/const identifier : type;

```
var x : int;
const pi : real = 3.14;
config const numSides : int = 4;
```

Serial Control Structures

- if statements, while loops, and do-while loops are all pretty standard
- Difference: Statement bodies must either use braces or an extra keyword:

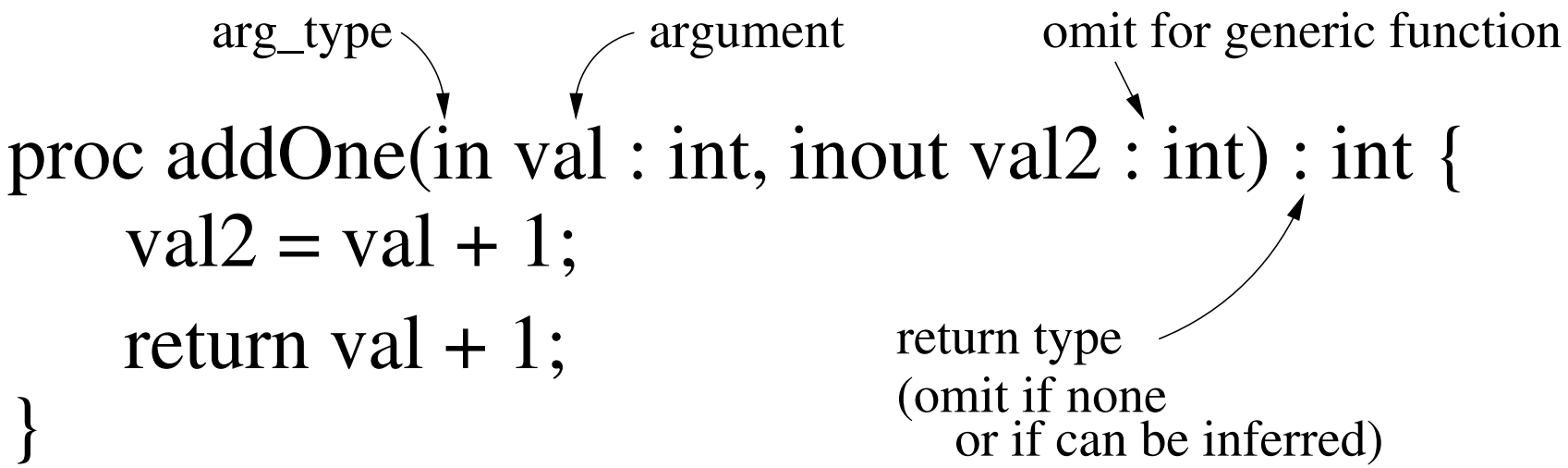
```
if(x == 5) then y = 3; else y = 1;

while(x < 5) do x++;
```

Example: Reading until eof

```
var x : int;
while stdin.read(x) {
    writeln("Read value ", x);
}
```

Procedures/Functions



Arrays

- Indices determined by a range:

```
var A : [1..5] int;              //declares A as array of 5 ints
var B : [-3..3] int;             //has indices -3 thru 3
var C : [1..10, 1..10] int;  //multi-dimensional array
```

- Accessing individual cells:

```
A[1] = A[2] + 23;
```

- Arrays have runtime bounds checking

For Loops

- Ranges also used in for loops:

```
for i in 1..10 do statement;
for i in 1..10 {
    loop body
}
```

- Can also use array or anything iterable

Timing code

```
use Time;                                   //include Time library

var timer = new Timer();                    //create Timer object

timer.start();
//do something...
timer.stop();

timer.elapsed()    //returns (real-valued) number of seconds
timer.clear();     //get ready to use it again!
```