Flexible Syntax

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

Why Chapell?

Basic Facts about Chapell

- High-level parallel programming
- Edited by J. Leheny, College of the Holy Cross, 2013
- Required reading for the National Science Foundation's summer course 'Parallel Programming' and 'Parallel Processing'
- Chapell is a high-level parallel programming language developed by UCSD and supported by the National Science Foundation
- Easy to use and learn
- Can be used in various settings, including Linux and Mac OS X

Schedule

- Summarize discussion
- Hands on time
- Parallel programming
- Parallel programming exercises
- Part II: 10:45-1:45 PM
  - Hands on time
  - Algorithms
  - Why Chapell
  - Part I: 10:30-10:45 AM

Acknowledgments

- Material drawn from literature created with contributions from various sources

Using Chapell

- Cygwin to install on Windows
- Easy to install on Linux and Mac OS X
- Suitable for shared or distributed memory systems
- Productivity/compiling system languages
- Originally developed at UCSD, high with Chapell
- Parallel programming language developed
Get handout from one of the instructors
During the workshop
We have practice accounts set up for use

(Accessing Practice Systems
during SC only)

Designed with Parallelism in Mind

Materials (on sourceforge)
http://chapel.sourceforge

Chapel website (tutorial, papers, language specification)
http://chapel.sourceforge

http://chapel.sourceforge

Our tutorials
http://heterogeneous-education/chapel/chapel

Materials for this workshop

Chapel Resources

Provides high-level operations (2)

... includes built-in operators:

\begin{align*}
E &= A \cdot B; \\
D &= A + B; \\
C &= A + 1;
\end{align*}

Function promotion:

\begin{align*}
b &= f(A); //\text{apply } f \text{ to elements for any function } f
\end{align*}

Not connected to Chapel development team

Educators who use Chapel with students

Interested in high-level parallel programming

Enthusiastic Chapel users

Your presenters are...

Provides high-level operations

Like a reduction, but computes value for each prefix

Scan

Ex: \(X = \text{reduce } A \text{ to sum of elements of } A\)

Ex: \(X = \text{reduce } A \text{ to sum of elements of } A\)
Serial Control Structures

Variables and Constants

Installing Chapel Yourself

Covering

Teach what you need - Chapel is not language

To Chapel syntax and provide tutorials

Give students a quick ("lectute) introduction

Using Chapel in Algorithms

Algorithms:

Easy implementation of parallelism

Hints.

For multiple installations (e.g. in /usr/local):

set environment variables

source /etc/chapel.sh or etc/chapel.sh

Enter Chapel T.8 directory and invoke make

Install:

Download:

http://chapellab.github.io/chapel/download/chapeltutorial.html

http://toolsandteaching.caltech.edu/teaching/tools/chapeltutorial.html

Installs chapel yourself
Arrays

Arrays have runtime bounds checking

```
var c : int[1..10] = [0] + [2];
```

Accessing individual cells:

```
for i = 0 to 3:
    print c[i];
```

Arrays defined by a range:

```
var a : int[1..5] = [0];
var b : int[3..5] = [0];
```

Indeces determined by a range:

```
var A : int[3..10, 1..20] = [0];
```

Two kinds of parallel loops:

- A thread that runs over a loop in parallel to another thread running over a different loop.
- A thread that runs over a single loop in parallel to another thread running over a different loop.

Procedure/Functions

Example: Reading until end of file

```
while stdin.read(x) != "":
    x = stdin.read(x)
```

For loops

```
{ loop body }
for i in 1..10
    for j in 1..10
        // do work
```

Ranges also used in for loops:

Sets can also use array of anything iterable

```
{ loop body }
for i in 1..10
    for j in 1..10
        // do work
```

Ranges also used in for loops:

```
{ loop body }
for i in 1..10
    for j in 1..10
        // do work
```

Asynchronous Tasks

```
// creates task per statement and waits here ...
begin { 
    statement;
    statement;
} 
```

Easy fork-join parallelism:

```
begin statement;
    // do work
end
```

Easy asynchronous task creation:

```
omit for generic function proc addOne(in val : int, inout val2 : int) : int {
return val + 1;
val2 = val + 1;
return type (omit if none or if can be inferred)
arg_type argument

```

Parallel loops

- A thread that runs over a loop in parallel to another thread running over a different loop.
- A thread that runs over a single loop in parallel to another thread running over a different loop.
Algorithms Project: BubbleSort

```c
{ compare ZK+1 to ZK+2 (maybe swap)
  for i in 0..n/2
    { compare ZK to ZK+1 (maybe swap)
      for i in 0..n/2
        { // don't know p vs NP yet
          brute-force algorithm
          // what are longest lists you can test?
          // explore
          // choices:
          // - nearest neighbors
          // - mergesort
          // - bubbleSort
          // - partition integers
          // - projects
          // - Teach Formal GRP in (also algorithmic notation)
          // - Assing basic tutorial
          // - change material
          // - sync blocks
          // - These are equivalent:
          // sync blocks exist for tasks created inside it
        }
    }
}
```

Algorithms Project: List Partition

```c
{ compare ZK+1 to ZK+2 (maybe swap)
  for i in 0..n/2
    { compare ZK to ZK+1 (maybe swap)
      for i in 0..n/2
        { // don't know p vs NP yet
          brute-force algorithm
          // what are longest lists you can test?
          // explore
          // choices:
          // - nearest neighbors
          // - mergesort
          // - bubbleSort
          // - partition integers
          // - projects
          // - Teach Formal GRP in (also algorithmic notation)
          // - Assing basic tutorial
          // - change material
          // - sync blocks
          // - These are equivalent:
          // sync blocks exist for tasks created inside it
        }
    }
}
```
Algorithms: Reductions

the brute-force method
Value of parallelism: much easier to program

• A bit tricky
• (use cothread)
– Divide-and-Conquer
– Brute force

Two algorithms:
• Find closest pair of (2-D) points.

Algorithms Project: Nearest Neighbors

Summing values in an array

Algorithms Project: MergeSort

Parallel divide-and-conquer: use cothread

0 1 4 4 8 9 12 15 16 4 5 7 9 16
4 5 7 9 16 4 5 7 9 16
0 1 4 4 8 9 12 15 16 4 5 7 9 16
4 5 7 9 16 4 5 7 9 16
0 1 4 4 8 9 12 15 16 4 5 7 9 16
4 5 7 9 16 4 5 7 9 16
0 1 4 4 8 9 12 15 16 4 5 7 9 16
4 5 7 9 16 4 5 7 9 16
0 1 4 4 8 9 12 15 16 4 5 7 9 16
4 5 7 9 16 4 5 7 9 16
Reduce 8n: Generate result from tally
Combine: Combine 2 tallies
Tally: Intermediate state of computation

Parts of a reduction

Finding max of an array

Finding the maximum index

Summing values in an array

Finding the maximum index

Summing values in an array
Sample problems:
- Accumulate: Add 1 value to tally
- Init: Create "empty" tally
- Reduce-Gen: Generate result from tally
- Combine: Combine 2 tallies
- Tally: Intermediate state of computation

Defining reductions

Two Issues

Parallel Reduction Framework

Parts of a Reduction

Return the Index
- Reduce-Gen: Generate result from tally
- Combine: Combine 2 tallies
- Take whichever pair has larger value

Two Issues

Parts of a Reduction
Relationship to dynamic programming

Sample problems + histogram, max
- Accumulate: Add 1 value to tally
- Initial: Create "empty" tally
- Reduce-Gen: Generate result from tally
- Combine: Combine 2 tallies
- Tally: Intermediate state of computation

Defining reductions

Sample problems + histogram, min
- Accumulate: Add 1 value to tally
- Initial: Create "empty" tally
- Reduce-Gen: Generate result from tally
- Combine: Combine 2 tallies
- Tally: Intermediate state of computation

Defining reductions

Can go beyond these...
Example: custom reduction

```java
// Define a custom type
enum Color { RED, BLUE, GREEN };

// Define a reduction function
public static int customReduce(int a, int b) {
    return a + b;
}
```

Inheritance

```java
class Shape {
    // Define a method to calculate area
    public int area() {
        return 0;
    }
}
```

Reduction example

```java
// Define a class to represent a shape
class Shape {
    public Shape() {
        // Define a method to calculate area
        public int area() {
            return 0;
        }
    }
}
```

Reductions in Chapel

```chapel
class Shape {
    // Define a method to calculate area
    public function area(): int {
        return 0;
    }
}
```

Reducing complexity: parallelism, composition, and other optimizations

- **General:** Converts state object into final output
- **Combine:** Adds another intermediate state
- **Accumulate:** Adds a single element to the state

- Must support
  - Create object to represent intermediate state

- Provides a framework for expressing and optimizing reduction operations in single line.
Defining your own reductions

- Optionally including scans
- Using reductions with standard functions

Computing the scan in parallel

- Upward pass to compute reduction
- Downward pass to compute reduction

Input:

```
1 2 3 4 5 6 7 8 9 10
```

Output:

```
10 11 12 13 14 15 16 17 18 19
```

And that's not all... (scans)
Now, let's use another thread:

```
void thread2() {
    for (int i = 0; i < 100; i++) {
        // do some work...
    }
}
```

We can add a timer to measure running time:

```
void timer() {
    long total;
    for (int i = 0; i < 100; i++) {
        // do some work...
        total += i;
    }
    // print total time...
}
```

This demonstrates that PL: Task Generation.

---

**High-Performance Computing as Programming Languages**

- Reduce overheads by leveraging thread pools
- Thread safety using variables of type `volatile`
- Parallel loops: parallel and scalable
- Task synchronization with `sync` and `cobegin`
- Task creation (instead of threads) with `begin`
- Lots of design choices in Chapel to parallelism

---

First hands on time
forall data-parallel loop

forall: forall

PL: Synchronousization

Chapel provides a solution: sync

incoherent: top thread may not finish

PL: Synchronousization

result: false, but sometimes incorrect:

-so, what did language designers do?

Ask students: how common is this?

PL: Synchronous Sugar

PL: Task Generation

PL: Synchronous Sugar
Students love it!

Learning HPC Motivates Syntax

Lots of language features to discuss.

PL: Takeaways

- How to synchronize?
  - Generate lots of tasks (corollary)
  - Collect completion testing

  - One entity at a time
  - Column-by-column
  - Different algorithms:
    - Matrix-vector multiplication
    - Matrix multiplication

PL: Projects

- Simplify with `reduce` keyword
- Divide-and-conquer solution

  - Reductions:
    - Running time still technically linear
    - Synchronization bottleneck and reduce

One solution: synchronized variables

Why doesn’t it work?

PL: HPC ConcepS

Ask: Why doesn’t this work?

Fork:

```
writeout("sum = ", sum);
for x in 1..100
  if x == 1
    sum = 0;

var sum : sync int;
```
Chapter Domains

- Are they cool operations?
- How are they used?
- A range of values
- What is a range?

Other Cool Range Things

- Otherwise, negations are just fine.

var nothing: range = 2..2;
Range in the "wrong order" are auto-empty:

var naturals: range = 0..;
Can create "infinite" ranges:

Range Operation Examples

-chaned ranges = [3, 5, 7, 9, 11]
var naturals: range = 0..<
var somedds: range = 0..49
var eveneds: range = 0..98
```javascript
{ // do something with A[i]
    for (i in A)
        ...
}
```

```javascript
var A: [1..10] int; // indices are 1, 2, ... 10
```

- Example:
  - can include ranges or be sparse
  - Every array has a domain to hold its indices
  - Used to simplify addressing
  - Domain: index set
Domain Slices (Intersection)

Domain: \{1.2, 3.3\}

Domain: \{1.3, 3.3\}

Domain: \{0.2, 1.3\}

Intersection

Integers are more general; some are not sets of

Domains are not sets.

Define a new domain with \( D \) as domain:

\[
\begin{align*}
D &= \{\text{domain}\} = \{0, 0.02\} \\
\text{Var} A : \text{Int} \\
\text{Var} B : \text{Domain} \\
\text{Var} A : [0, 0.02] \\
\text{Var} B : \{0, 0.02\} \\
\text{Intersection} = \{0, 0.02\} \\
\text{Domain} = \{0, 0.02\}
\end{align*}
\]

Domain Slices (Intersection)

Domain: \{1.2, 3.3\}

Domain: \{1.3, 3.3\}

Domain: \{0.2, 1.3\}

Chapel Domains

Why is this useful?

- A is an array of integers with \( d \) as domain
- \( d \) is the domain with \( m+1 \) entries
- Var A: [0, \( m \)]
- Var B: Domain = \( (0, 0.02) \)
- Domain declaration:

   - Domain: \( \{0, 0.02\} \)

"Chapel Domains"
**Unbounded?**

**Domain:** Unbounded Game of Life

- Pad all sides with zeros.
- Start with an empty board.
- Plan: board starts with small living area, but can grow.

**Rules:**
- Each cell is either dead or alive.
- Dead cell
- Living cell
- Dead cell
- Living cell
- Adjacent to 8 surrounding cells
- Each cell is either dead or alive.
- Changing domain for arrays
- One domain for multiple arrays
- Domain operations
- Example of

```javascript
var domains = domain1 & domain2;
// domain2 is the intersection of domain1 and domain2
```

**Unbounded?**

**Domain:** Unbounded Slices (Intersection)

- Start with an empty board.
- Plan: board starts with small living area, but can grow.

```javascript
domains: {1, 2, 3, 4};
domains: {1, 3, 4};
domains: {2, 3, 4};
```
Game of Life: Setting the Domain

Unbounded How?

Repeat –

(any) –

Reach edge & draw with living cells

Select one round

Feed all sides with zeros

Start with dead board

Plan board starts with small living area, but can grow?

Plan board starts with small living area, but can grow?
Game of Life: Implementing Rules
Game of Life: Supporting Boards

Game of Life: Implementing Rules

Game of Life: Implementing Rules

Game of Life: Implementing Rules
Game of Life: "If Alive" Functions

Please provide the content of the image so I can assist you better.
Game of life in MPI

My experience

Parallel Programming

Game of life: Main loop

| Taked about Chapel (and ZPL) in contrast to (apps, median, system software, programming) |
| Course to explore HP overall |

Game of life: Add writing and go!

Final version available at:

- Add a function
  - Loop and watch for 10
    - Print statements for each iteration of the loop

- My experience

Separate from parallelism

Global-view

Representing Locality

Much harder than I thought

Game of life in MPI
• Independent Projects
  (or other courses w/ computational-intense
  Artificial Intelligence
  Chapter Implementations
  – Some parallel design patterns have inheritance
  Software Design
  – Easy thread generation for scheduling projects
  Operating Systems

How else might you use Chapex?

Summary / Discussion

Useful References

var a: [d] in mapped Cyclical(standardx=1)

var a: [d] in mapped Block(boundingbox=d)

Domains map by how arrays are mapped

Managing data distribution

Take Home: Parallel course
Your Feedback

- Why was a bunch and is happy to share?
- What resources would help you adopt it?
- What course(s) will you use it in?
- How likely are you to adopt Chapter?
- What are your impressions of Chapter?

Conclusions

- Students will dig in
- Flexible depth of material
- Loads of features, but...
- Chapter can be used in many courses
- Chapter is easy to pick up

Caveats

- Command-line compilation in Linux
- No development environment
- (Students thought this was awesome)
- Not many libraries
- New versions every 6 months
- Error messages thin
- Still in development

Thanks!