What is Chapel?

“Chapel is a modern parallel programming language designed for productivity at scale.”
What Does That Mean?

- Pay attention to the issues in large-scale parallel programming
  - Control
  - Memory

- But have features that look like a “real” programming language
  - Types, type inference, objects, …
History

• Main paper in 2007
  • Read the intro to this paper!

• Preceded by Cascade
  • ~2004
  • And by ZPL before that ...

• Still an active project today
Model of Control

• In a word: threads
A Few Words About Threads

• A thread is a sequential program

• Multiple threads can execute in parallel
  • All statements in a single thread execute in the specified order
  • There is no specified ordering of instructions in different threads
  • Instructions from different threads may be interleaved in any order

• And threads share state
  • One thread can write a value another thread reads
Example 1

Thread 1

\[ x = 0 \]
\[ y = 0 \]
\[ x = x + 1 \]

Thread 2

\[ x = 0 \]
\[ y = 0 \]
\[ y = y + 1 \]
Example 2

Thread 1

x = 0
y = 0
x = y + 1

Thread 2

x = 0
y = 0
y = y + 1
Example 3

Thread 1
initially $y = 0$

$y = y + 1$

Thread 2

$y = y + 1$
Example 3: Machine Level

Thread 1
initially y = 0

r1 = load y
r1 = r1 + 1
y = store r1

Thread 2

r2 = load y
r2 = r2 + 1
y = store r2
Example 3: Atomics

Thread 1
initially $y = 0$

atomic\{ $y = y + 1$ \}

Thread 2

atomic\{ $y = y + 1$ \}
Thread Synchronization

• Threading systems often have a wide array of synchronization primitives
  • Ways to restrict the interleavings of threads

• General philosophy
  • Allow any interleavings by default
  • Add enough synchronization to eliminate undesirable interleavings
Data Parallelism in Chapel

• Index domains, both structured and unstructured

• Parallel for loops

  \[
  \text{forall } i \text{ in } I \text{ do } ...
  \]

• Legion/Regent use Chapel-style domains
Task Parallelism

- `cobegin {s1; s2}`

- Statements `s1` and `s2` may run in parallel

- Structured future-like variables for inter-thread communication
  - Variables can be either `full` or `empty`
  - A write fills the variable
  - A read empties it
  - Producer-consumer style parallelism
Nested Parallelism

• Constructs can be arbitrarily nested

• Fine to have
  • Task parallelism inside of data parallelism
  • Or vice versa
Reductions

• Built-in support for reductions and scans.

• Not integrated directly with other features
  • Really a separate facility
  • But can be used in combination with other kinds of parallelism
Locales

• *Locales* name places where computation can happen and values can be stored

• Locales are an abstract concept
  • In practice, a node would be a locale

• Note: The set of all locales is just ... a set
  • No structure
  • No topological relationships between set elements
  • Combines processors and memories in one
Data Model: Distributions

• A domain can be *distributed* among a set of locales

• Chapel supports standard distributions
  • Blocked, cyclic, blocked cyclic
  • And user-defined distributions
Alignment

• A new Chapel distribution can be defined as an alignment with an existing distribution

• E.g., “Layout index set B like index set A”

• Allows distributions to be derived from existing distributions
  • Compare with Legion/Regent’s dependent partitioning
Owner Computes

• Consider \texttt{forall i in I do ...}

• Default execution uses the \textit{owner computes} rule:
  • Iteration \texttt{i} is executed on the locale that owns it

• Programmer can override:
  \texttt{forall i in I do on A[i+1] do ...}
User-Defined Distributions

• Additional distributions can be defined

• Implement distribution interface
  • A lower-level API for defining distributions

• The standard distributions are also written this way
  • But there is a difference in compiler knowledge!
Example

\[
\text{const indices} = \{1..1000\} \text{ dmapped Cyclic(startIdx} = 1)\\
\text{forall i in indices do}\\
\text{ writeln(“iteration \text{“}, i, “on locale \text{“}, here.id)}
\]
Object-Oriented Features

• Chapel strives to look and feel like a modern object-oriented language
  • E.g., Java

• But not fully OO
  • Emphasis on arrays and pass-by-value
  • Because of importance in high-performance computing
Chapel Critique

• Machine Model

• Memory/Data

• Control

• Latency hiding
Machine Model

• Designed for a world of clusters

• Locales are essentially a flat collection
  • Fine if the locales are nodes that are peers on a network
  • Reality is now more complex due to accelerators and other heterogeneity within a node
    • e.g., NUMA
Memory/Data Model

• Model of machine memory is very simple

• Unified with computation
  • No mechanism for talking about different memories accessible from the same processor
  • No mechanism for talking about hierarchy
Data Model

• Lots of support for manipulating, partitioning index spaces
  • A good idea, widely adopted
  • Can also define and use subspaces
  • Sparse index space support is not fully worked out

• Note: The index space itself is mapped, not the data
  • Allows multiple arrays with the same index space to be trivially partitioned the same way
Data Model (Cont.)

• Emphasis on where the data is placed
  • And can only have one placement

• No (?) facilities for expressing movement of data
  • Data movement is implicit, in that if a thread on a locale needs a value from another locale the needed messages are generated automatically
  • Data movement at the granularity of individual requests
Control

• Locales are virtualized processors
  • And memories

• Fine to have multiple threads per processor
  • No guarantee of exclusive access

• Ability to run multiple threads/locale is also the latency hiding mechanism

• Various synchronization mechanisms we didn’t discuss
  • Present in all threading models
X10

• Surprisingly similar to Chapel!

• Not really fair ...
  • There are real differences in the designs
  • But not so much at the level of today’s lecture

• X10 is
  • Thread-based
  • Provides data parallel and task parallel constructs
  • Has a flat model of compute/memory locations called *places*
X10: What’s Different

• Java-based
  • More emphasis on integration into an existing language
  • More emphasis on being object-oriented

• Garbage-collected
  • A huge difference
  • Only serious HPC effort that uses GC
  • Every local JVM collects its own heap on a node
  • Make sure inter-node references are tracked
    • So that data pointed to on remote nodes isn’t collected!