Chapel

The Cascade High-Productivity Language

PGAS

Chapel Motivation

Overview

Chapel: a new parallel programming language being developed by Cray Inc. as part of DARPA's High *Productivity* Computing Systems program (HPCS)

Motivating Themes:

- **general parallel programming**
  - to support data-, task-, and nested parallelism
  - to support general granularities of software parallelism
  - to target general levels of hardware parallelism
- ***global-view* abstractions**
  - to reduce the number of programmer-managed details
- **multiresolution design**
  - to support abstraction without giving up low-level tuning
- **reduce mainstream-to-parallel language gap**
  - to engage the next generation of programmers
  - to take advantage of advances in language design

Productivity

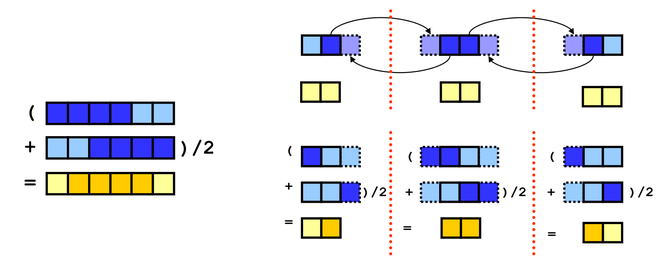
HPCS Goal: Raise user productivity by 10x

Chapel's productivity goals:

- Vastly improve **programmability**
  - writing parallel codes
  - reading, modifying, porting, tuning, maintaining them
- Support **performance** at least as good as MPI
  - competitive with MPI on generic clusters
  - better than MPI on more capable architectures
- Improve **portability**
  - as ubiquitous as MPI, but with fewer ties to architecture
  - more portable than other parallel programming models
- Improve **code robustness**
  - eliminate common error cases
  - support better abstractions to help avoid other errors

Global View of Data and Control

Example: "Apply a 3-point stencil to a vector"



global-view

```
def main() {
  var n: int = 1000;
  var a, b: [1..n] real;

  forall i in 2..n-1 {
    b[i] = (a[i-1] + a[i+1])/2;
  }
}
```

local-view/SPMD

```
def main() {
  var n: int = 1000;
  var locN: int = n/numProcs;
  var a, b: [0..locN+1] real;
  var innerLo: int = 1;
  var innerHi: int = locN;

  if (iHaveRightNeighbor) {
    send(right, a[locN]);
    recv(right, a[locN+1]);
  } else {
    innerHi = locN-1;
  }
  if (iHaveLeftNeighbor) {
    send(left, a[1]);
    recv(left, a[0]);
  } else {
    innerLo = 2;
  }
  forall i in innerLo..innerHi {
    b[i] = (a[i-1] + a[i+1])/2;
  }
}
```

Chapel supports a global view of data + control flow for productivity and a local view for low-level control

Language Features

Task Parallelism

begin: creates a task for concurrent evaluation

```
begin DoThisTask();
WhileContinuing();
TheOriginalThread();
```

sync: waits on all tasks created within its dynamic scope

```
sync {
  begin treeSearch(root);
}

def treeSearch(node) {
  if node == nil then return;
  begin treeSearch(node.right);
  begin treeSearch(node.left);
}
```

sync/single variables: store full/empty state with their value

```
var result$: sync real;    // result is initially empty
sync {
  begin … = result$;       // block until full, leave empty
  begin result$ = …;       // block until empty, leave full
}
result$.readXX();          // non-blocking, leaves state unchanged;
                           // other variations also supported
```

cobegin statement: creates a task per component statement

```
computePivot(lo, hi, data);
cobegin {
  Quicksort(lo, pivot, data);
  Quicksort(pivot, hi, data);
} // implicit join here
```

```
cobegin {
  computeTaskA(…);
  computeTaskB(…);
  computeTaskC(…);
} // implicit join here
```

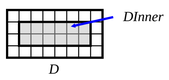
coforall loop: creates a task per loop iteration

```
coforall e in Edges {
  exploreEdge(e);
} // implicit join here
```

Data Parallelism

domain: a first-class index set

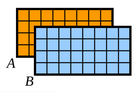
```
const m = 4, n = 8;
const D: domain(2) = [1..m, 1..n];
var Inner: subdomain(D) = [2..m-1, 2..n-1];
```



Domain uses:

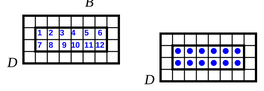
- Declaring arrays:

```
var A, B: [D] real;
```



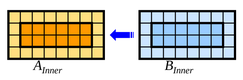
- Iteration (sequential or parallel):

```
for     ij in Inner { … }
forall ij in Inner { … }
```



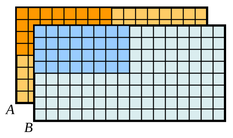
- Array Slicing:

```
A[Inner] = B[Inner];
```

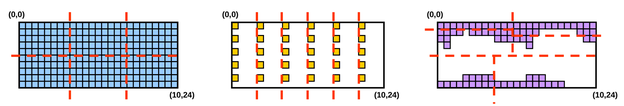


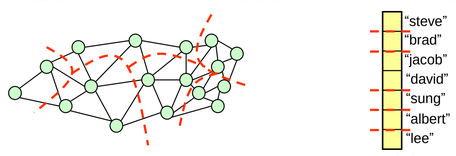
- Array reallocation:

```
D = [1..2*m, 1..2*n];
```



Domain types:





Locality

locale: architectural unit of locality

- has capacity for processing and storage
- threads within a locale have ~uniform access to local memory
- memory within other locales is accessible, but at a price
- *e.g.*, a multicore processor or SMP node could be a locale

- user specifies # locales on executable command-line

```
prompt> myChapelProg -nl=8
```

- Chapel programs have built-in locale variables:

```
config const numLocales: int;
const LocaleSpace = [0..numLocales-1],
      Locales: [LocaleSpace] locale;
```

L0 L1 L2 L3 L4 L5 L6 L7

- Programmers can create their own locale views:

```
var CompGrid = Locales.reshape([1..GridRows, 1..GridCols]);
```

L0 L1 L2 L3
L4 L5 L6 L7

```
var TaskALocs = Locales[..numTaskALocs];
var TaskBLocs = Locales[numTaskALocs+1..];
```

L0 L1

L2 L3 L4 L5 L6 L7

- *on clauses:* indicate where tasks should execute

```
cobegin {
  on TaskALocs do computeTaskA(…);
  on TaskBLocs do computeTaskB(…);
  on Locales(0) do computeTaskC(…);
}
```

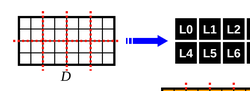
L0 L1 computeTaskA()

L2 L3 L4 L5 L6 L7 computeTaskB()

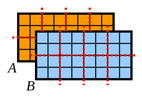
L0 computeTaskC()

- Domains may be mapped across locales

```
var D: domain(2) dmapped Block(…) = …;
```



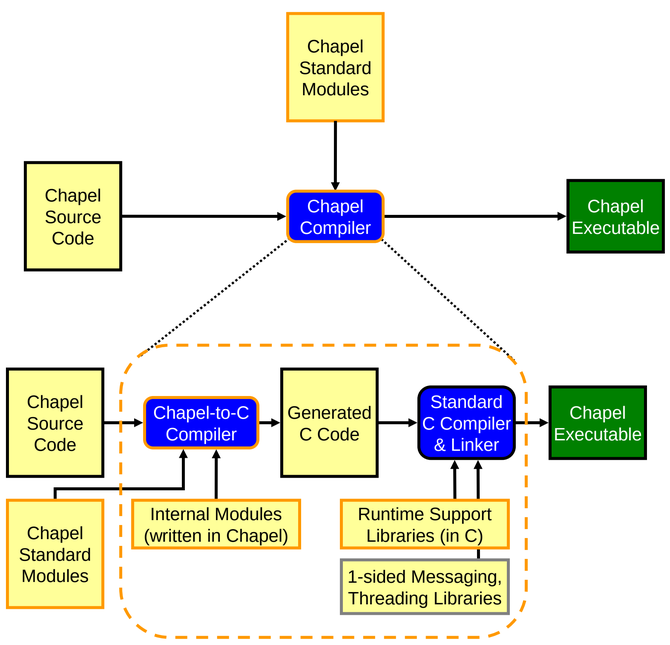
- A domain map defines…
  - …ownership of the domain's indices (and its arrays' elements)
  - …the default work ownership for operations on the domains/arrays



- Chapel provides…
  - …a standard library of distributions (Block, Block-Cyclic, Recursive Bisection, …)
  - …the means for advanced users to write their own distributions

Implementation Status

Prototype Compiler



Public Release

- **Latest release:** version 1.2 (October 21, 2010)
- **Supported platforms:**
  - desktop Linux/UNIX, Mac OS X, Cygwin, …
  - commodity clusters of same
  - Cray platforms
  - Systems from other vendors
- Download from: http://sourceforge.net/projects/chapel
- Your feedback desired!
- Keep in mind this is a work-in-progress:
  - ⇒ it's likely that you will find problems with the implementation
  - ⇒ this is still a reasonable time to influence the language

- **Base language:** stable, yet incomplete: I/O, OOP, exceptions
- **Task parallelism:**
  - multi-threaded, multi-locale implementation of tasks, sync variables
- **Data parallelism:**
  - multi-threaded, multi-locale regular domains/arrays
  - multi-threaded, single-locale irregular domains/arrays
- **Locality:**
  - locale types and arrays
  - richer set of distributions to be developed: Block-Cyclic, MRD, …
- **Performance:**
  - has received much attention in designing the language
  - work remains in communication optimizations; serial code

Notable Collaborations

Notre Dame/ORNL (Peter Kogge, Srinivas Sridharan, Jeff Vetter): Asynchronous STM over distributed memory

UIUC (David Padua, Maria Garzarán, Albert Sidelnik): Chapel for hybrid CPU-GPU computing

Universitat Politècnica de Catalunya (Alex Duran): Chapel over Nanos++ user-level tasking

Argonne (Rusty Lusk, Rajeev Thakur, Pavan Balaji): Chapel over MPICH one-sided communication

Sandia (Richard Murphy, Kyle Wheeler): Chapel over Qthreads user-level tasking

UT Austin (Calvin Lin, Karthik Murthy): Productive memory consistency models

CU Boulder (Jeremy Siek, Jonathan Turner): interfaces, concepts, and generics

U. Oregon/Paratools Inc. (Sameer Shende): Performance analysis with Tau

Interested in collaborating? See our website for possible topics; or propose your own.

DARPA HPCS

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0001. 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 Defense Advanced Research Projects Agency.

***For more information, visit:* http://chapel.cray.com *or mail:* chapel_info@cray.com**

CRAY