Chapel: Background

Steve Deitz
Cray Inc.
**HPCS: High Productivity Computing Systems (DARPA)**

- **Goal:** Raise HEC user productivity by 10x by 2010
  
  \[ \text{Productivity} = \text{Performance} + \text{Programmability} + \text{Portability} + \text{Robustness} \]

- **Phase II:** Cray, IBM, Sun (July 2003 – June 2006)
  - Evaluated entire system architecture
  - Three new languages (Chapel, X10, Fortress)

- **Phase III:** Cray, IBM (July 2006 – 2010)
  - Implement phase II systems
  - Work continues on all three languages
Chapel Productivity Goals

- Improve programmability over current languages
  - Writing parallel codes
  - Reading, changing, porting, tuning, maintaining, ...
- Support performance at least as good as MPI
  - Competitive with MPI on generic clusters
  - Better than MPI on more capable architectures
- Improve portability over current languages
  - As ubiquitous as MPI
  - More portable than OpenMP, UPC, CAF, ...
- Improve robustness via improved semantics
  - Eliminate common error cases
  - Provide better abstractions to help avoid other errors
Chapel Design Concepts

- Support general parallel programming
  - Data, task, and nested parallelism
  - Express all levels of software parallelism
  - Target all levels of hardware parallelism
- Support global-view abstractions
- Support multiple levels of design
- Allow for control of locality
- Bring mainstream features to parallel languages
Outline

- Concepts, Settings and Goals
- Chapel’s Programming Model
  - Fragmented vs. **Global-View**
  - Low-Level vs. High-Level vs. **Multiple Levels**
Fragmented vs. Global-View: Definitions

- **Programming model**
  *The mental model of a programmer*

- **Fragmented models**
  *Programmers take point-of-view of a single processor/thread*

- **SPMD models (Single Program, Multiple Data)**
  *Fragmented models with multiple copies of one program*

- **Global-view models**
  *Programmers write code to describe computation as a whole*
3-Point Stencil Example (n=6)

Initial state

<table>
<thead>
<tr>
<th></th>
<th>2.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>12.0</th>
</tr>
</thead>
</table>

Iteration 1

<table>
<thead>
<tr>
<th></th>
<th>2.0</th>
<th>1.0</th>
<th>0.0</th>
<th>0.0</th>
<th>6.0</th>
<th>12.0</th>
</tr>
</thead>
</table>

Iteration 2

<table>
<thead>
<tr>
<th></th>
<th>2.0</th>
<th>1.0</th>
<th>0.5</th>
<th>3.0</th>
<th>6.0</th>
<th>12.0</th>
</tr>
</thead>
</table>

Iteration 3

<table>
<thead>
<tr>
<th></th>
<th>2.0</th>
<th>1.25</th>
<th>2.0</th>
<th>3.25</th>
<th>7.5</th>
<th>12.0</th>
</tr>
</thead>
</table>

Steady state

|   | 2.0 | 4.0 | 6.0 | 8.0 | 10.0 | 12.0 |
Global-View vs. Fragmented Data Declarations

Global-View

Fragmented
Global-View vs. Fragmented Computation

**Global-View**

\[
\frac{\left( \begin{array}{cccc}
1 & 2 & 3 & 4
\end{array} \right)}{2} + \frac{\left( \begin{array}{cccc}
4 & 5 & 6 & 7
\end{array} \right)}{2} = \left( \begin{array}{cccc}
2 & 4 & 6 & 8
\end{array} \right)
\]

**Fragmented**

\[
\frac{\left( \begin{array}{cccc}
1 & 2 & 3 & 4
\end{array} \right)}{2} + \frac{\left( \begin{array}{cccc}
4 & 5 & 6 & 7
\end{array} \right)}{2} = \frac{\left( \begin{array}{cccc}
2 & 4 & 6 & 8
\end{array} \right)}{2} + \frac{\left( \begin{array}{cccc}
7 & 8 & 9 & 10
\end{array} \right)}{2} = \left( \begin{array}{cccc}
3 & 5 & 7 & 9
\end{array} \right)
\]

Chapel: Background
Global-View vs. Fragmented Code

**Global-View**

```chapel
def main() {
    var n = 1000;
    var A, B: [1..n] real;
    forall i in 2..n-1 do
        B(i) = (A(i-1)+A(i+1))/2;
}
```

**Fragmented**

```chapel
def main() {
    var n = 1000;
    var me = commRank(), p = commSize(),
        myN = n/p, myLo = 1, myHi = myN;
    var A, B: [0..myN+1] real;
    if me < p {
        send(me+1, A(myN));
        recv(me+1, A(myN+1));
    } else myHi = myN-1;
    if me > 1 {
        send(me-1, A(1));
        recv(me-1, A(0));
    } else myLo = 2;
    for i in myLo..myHi do
        B(i) = (A(i-1)+A(i+1))/2;
}
```

Assumes p divides n
NAS MG Stencil

\[ w_0 + w_1 + w_2 + w_3 \]
use caf_intrinsics

implicit none

include 'cafmsg.h'
include 'globals.h'
integer n1, n2, n3, k
double precision u(n1,n2,n3)
integer axis
if( .not. dead(kk) ) then
do axis = 1, 3
if( axis .eq. 1 ) then
call givev( axis, 1, n1, n2, k, 1, 0)
call givev( axis, -1, n1, n2, n3, 0, 0)
call takev( axis, -1, n1, n2, n3, 0, 0)
call takev( axis, 1, n1, n2, n3, 0, 0)
call comep( axis, u, n1, n2, n3)
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endid
def rprj3(S, R) {
const Stencil = [-1..1, -1..1, -1..1],
    W: [0..3] real = (0.5, 0.25, 0.125, 0.0625),
    W3D = [(i,j,k) in Stencil] W((i!=0)+(j!=0)+(k!=0));

forall inds in S.domain do
    S(inds) =
        + reduce [offset in Stencil] (W3D(offset) * 
            R(inds + offset*R.stride));
}

Previous work shows performance is still possible:

## Summary of Current Programming Systems

<table>
<thead>
<tr>
<th>System</th>
<th>Data Model</th>
<th>Compute Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Communication Libraries</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MPI/MPI-2</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td>SHMEM</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td>ARMCI</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td>GASNet</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td><strong>Shared Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP, pThreads</td>
<td><strong>Global-View (trivially)</strong></td>
<td><strong>Global-View (trivially)</strong></td>
</tr>
<tr>
<td><strong>PGAS Languages</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Co-Array Fortran</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td>UPC</td>
<td><strong>Global-View</strong></td>
<td>Fragmented</td>
</tr>
<tr>
<td>Titanium</td>
<td>Fragmented</td>
<td>Fragmented</td>
</tr>
<tr>
<td><strong>HPCS Languages</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapel</td>
<td><strong>Global-View</strong></td>
<td><strong>Global-View</strong></td>
</tr>
<tr>
<td>X10 (IBM)</td>
<td><strong>Global-View</strong></td>
<td><strong>Global-View</strong></td>
</tr>
<tr>
<td>Fortress (Sun)</td>
<td><strong>Global-View</strong></td>
<td><strong>Global-View</strong></td>
</tr>
</tbody>
</table>
Low-Level vs. High-Level Programming

```
“Why is everything so difficult?”
```

```
“Why can’t I optimize this?”
```
Multiple Levels of Design

**Task Scheduling**
- Work Stealing
- Suspended Tasks
- Task Pool
- Thread per Task
- Target Machine

**Language Concepts**
- Data Parallelism
- Distributions
- Task Parallelism
- Base Language
- Locality Control
- Target Machine

**Memory Management**
- Garbage Collection
- Region-Based
- Malloc/Free
- Target Machine

Chapel: Background
Questions?

- Concepts, Settings and Goals
- Chapel’s Programming Model
  - Fragmented vs. Global-View
  - Low-Level vs. High-Level vs. Multiple Levels