Chapel
Cray Cascade’s High Productivity Language

Mary Beth Hribar
Steven Deitz
Brad Chamberlain

Cray Inc.

CUG 2006
Chapel Contributors

- **Cray Inc.**
  - Brad Chamberlain
  - Steven Deitz
  - Shannon Hoffswell
  - John Plevyak
  - Wayne Wong
  - David Callahan
  - Mackale Joyner

- **Caltech/JPL:**
  - Hans Zima
  - Roxana Diaconescu
  - Mark James
Chapel’s Context

**HPCS** = High *Productivity* Computing Systems  
(a DARPA program)

**Overall Goal:** Increase productivity by $10 \times$ by 2010

**Productivity** = Programmability  
+ Performance  
+ Portability  
+ Robustness

**Result must be...**  
...revolutionary, not evolutionary  
...marketable product

**Phase II Competitors (7/03-7/06):** Cray (Cascade), IBM, Sun
Chapel Design Objectives

- a *global view* of computation
- support for general parallelism
  - data- and task-parallel; nested parallelism
- clean separation of algorithm and implementation
- broad-market language features
  - OOP, GC, latent types, overloading, generic functions/types, …
- data abstractions
  - sparse arrays, hash tables, sets, graphs, …
- good performance
- portability
- interoperability with existing codes
Outline

- Chapel Motivation & Foundations
  - Context and objectives for Chapel
  - Programming models and productivity
- Chapel Overview
- Chapel Activities and Plans
Parallel Programming Models

• **Fragmented Programming Models:**
  - Programmers *must* program on a task-by-task basis:
    - break distributed data structures into per-task chunks:
    - break work into per-task iterations/control flow

• **Global-view Programming Models:**
  - Programmers need not program task-by-task
    - access distributed data structures as though local
    - introduce parallelism using language keywords
    - burden of decomposition shifts to compiler/runtime
    - user may guide this process via language constructs
Global-view vs. Fragmented

- **Example:** “Apply 3-pt stencil to vector”

\[
\text{global-view} = \left( \frac{\text{vector1}}{2} + \text{vector2} \right) / 2
\]

This Presentation May Contain Some Preliminary Information, Subject To Change
Global-view vs. Fragmented

• **Example:** “Apply 3-pt stencil to vector”

\[
\begin{align*}
\text{global-view} & \quad \frac{1}{2} \\
( & \begin{array}{cccc}
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\end{array} + \begin{array}{cccc}
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\end{array} ) / 2 \\
= & \begin{array}{cccc}
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\text{ } & \text{ } & \text{ } & \text{ } \\
\end{array}
\end{align*}
\]
Global-view vs. Fragmented

- **Example:** “Apply 3-pt stencil to vector”

  \[
  \text{global-view} \quad \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right) = \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right) = \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right)\]

  \[
  \text{fragmented} \quad \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right) = \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right) = \frac{1}{2} \left( \frac{1}{2} + \frac{1}{2} \right)
  \]
Global-view vs. Fragmented

- **Example:** “Apply 3-pt stencil to vector”

```plaintext
global-view

var n: int = 1000;
var a, b: [1..n] float;

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

Assumes numProcs divides n; a more general version would require additional effort
```

```plaintext
fragmented

var n: int = 1000;
var locN: int = n/numProcs;
var a, b: [0..locN+1] float;
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;
}
```

This Presentation May Contain Some Preliminary Information, Subject To Change
Global-view vs. Fragmented

• Example: “Apply 3-pt stencil to vector”

fragmented (pseudocode + MPI)

```
var n: int = 1000, locN: int = n/numProcs;
var a, b: [0..locN+1] float;
var innerLo: int = 1, innerHi: int = locN;
var numProcs, myPE: int;
var retval: int;
var status: MPI_Status;

MPI_Comm_size(MPI_COMM_WORLD, &numProcs);
MPI_Comm_rank(MPI_COMM_WORLD, &myPE);
if (myPE < numProcs-1) {
    retval = MPI_Send(&(a(locN)), 1, MPI_FLOAT, myPE+1, 0, MPI_COMM_WORLD);
    if (retval != MPI_SUCCESS) { handleError(retval); }
    retval = MPI_Recv(&(a(locN+1)), 1, MPI_FLOAT, myPE+1, 1, MPI_COMM_WORLD, &status);
    if (retval != MPI_SUCCESS) { handleErrorWithStatus(retval, status); }
} else {
    innerHi = locN-1;
    if (myPE > 0) {
        retval = MPI_Send(&(a(1)), 1, MPI_FLOAT, myPE-1, 1, MPI_COMM_WORLD);
        if (retval != MPI_SUCCESS) { handleError(retval); }
        retval = MPI_Recv(&(a(0)), 1, MPI_FLOAT, myPE-1, 0, MPI_COMM_WORLD, &status);
        if (retval != MPI_SUCCESS) { handleErrorWithStatus(retval, status); }
    } else {
        innerLo = 2;
    }
}
forall i in (innerLo..innerHi) {
    b(i) = (a(i-1) + a(i+1))/2;
}
```

Communication becomes geometrically more complex for higher-dimensional arrays.
Fortran+MPI 3D NAS MGStencil

subroutine comm3(u,n1,n2,n3,kk)
  implicit none
  include 'globals.h'
  integer n1, n2, n3, kk
  double precision u(n1,n2,n3)
  if( axis .eq. 2 ) then
    else if( dir .eq. +1 ) then
      if( axis .eq. 3 ) then
        use caf_intrinsics
        do  i3=2,n3-1
          do  i1=1,n1
            buff_len = buff_len + 1
            indx = indx + 1
            buff(buff_len, buff_id) = u(i1,i2,n3-1)
          enddo
        enddo
      endif
      do  i3=2,n3-1
        do  i2=1,n2
          do  i1=1,n1
            buff_id = 2 + dir
            if( .not. dead(kk) ) then
              if( axis .eq. 3 ) then
                indx = indx + 1
              endif
              if( axis .eq. 2 ) then
                buff(1:buff_len,buff_id+1)(nbr(axis,dir,k)) =
                  buff(1:buff_len,buff_id)
              endif
              buff(i,buff_id) = 0.0D0
            enddo
          enddo
        enddo
      enddo
      rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k)
      call zero3(u,n1,n2,n3)
    else if( axis .eq. 3 ) then
      use caf_intrinsics
      do  i=1,nm2
        buff(i,4) = buff(i,3)
      enddo
      endif
      do  i=1,nm2
        buff(i,2) = buff(i,1)
      enddo
dir = +1
  endif
  endif
  dir = -1
  double precision x1(m), y1(m), x2,y2
  do  i3=2,n3-1
    do  i2=2,n2-1
      x1(i1-1) = r(i1-1,i2-1,i3) + r(i1-1,i2+1,i3)
      + r(i1-1,i2, i3-1) + r(i1-1,i2, i3+1)
      x2 = r(i1, i2-1,i3) + r(i1, i2+1,i3)
      + r(i1, i2, i3-1) + r(i1, i2, i3+1)
    enddo
  enddo
  endif
  return
endif
else if( axis .eq. 3 ) then
  if( axis .eq. 2 ) then
    do  i3=2,n3-1
      do  i2=1,n2
        do  i1=1,n1
          buff_id = 3 + dir
          indx = 0
          if( axis .eq. 3 ) then
            indx = indx + 1
          endif
          if( axis .eq. 2 ) then
            buff(1:buff_len,buff_id+1)(nbr(axis,dir,k)) =
              buff(1:buff_len,buff_id)
            endif
            buff(i,buff_id) = 0.0D0
          enddo
      enddo
    enddo
  endif
  endif
  do  i=1,nm2
    buff(i,3) = buff(i,2)
  enddo
dir = -1
  double precision x1(m), y1(m), x2,y2
  do  i3=2,n3-1
    do  i2=2,n2-1
      x1(i1-1) = r(i1-1,i2-1,i3) + r(i1-1,i2+1,i3)
      + r(i1-1,i2, i3-1) + r(i1-1,i2, i3+1)
      x2 = r(i1, i2-1,i3) + r(i1, i2+1,i3)
      + r(i1, i2, i3-1) + r(i1, i2, i3+1)
    enddo
  enddo
  endif
  return
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endif
endi
Chapel 3D NAS MG Stencil

```chapel
param coeff: domain(1) = [0..3]; // for 4 unique weight values
param Stencil: domain(3) = [-1..1, -1..1, -1..1]; // 27-points

function rprj3(S, R) {
    param w: [coeff] float = (/0.5, 0.25, 0.125, 0.0625/);
    param w3d: [(i,j,k) in Stencil] float = w((i!=0) + (j!=0) + (k!=0));
    const SD = S.domain,
               Rstr = R.stride;

    S = [ijk in SD] sum reduce [off in Stencil] (w3d(off) * R(ijk + Rstr*off));
}
```

This Presentation May Contain Some Preliminary Information, Subject To Change
Fragmented Language Summary

- Fragmented programming models…
  …manage per-task details in-line with the computation
  - per-task local bounds, data structures
  - communication, synchronization
  …are our main parallel programmability limiter today
Fragmented Language Summary

- Fragmented programming models…
  …tend to be easier to compile than global-view languages
  • at minimum, only need a good node compiler
  …deserve credit for the majority of the community’s parallel application successes to date
Global-View Language Summary

- Single-processor languages are trivially global-view
  - Matlab, Java, Python, Perl, C, C++, Fortran, ...

- Parallel global-view languages have been developed...
  - HPF (High Performance Fortran), ZPL, Sisal, NESL, Cilk, Cray MTA extensions to C/Fortran, ...

- ...yet most have not achieved widespread adoption
  - reasons why are as varied as the languages themselves

- Chapel has been designed...
  ...to support global-view programming
  ...with experience from preceding global-view languages
Outline

- Chapel Motivation & Foundations
- Chapel Overview
  - Chapel Activities and Plans
What is Chapel?

- **Chapel**: Cascade High-Productivity Language
- **Overall goal**: “Solve the parallel programming problem”
  - simplify the creation of parallel programs
  - support their evolution to extreme-performance, production-grade codes
  - emphasize generality
- **Motivating Language Technologies**:
  - global-view multithreaded parallel programming
  - locality-aware programming
Multithreaded Parallel Programming

- Virtualization of threads
  - *i.e.*, no fork/join, naming of threads

- Abstractions for data and task parallelism
  - *data*: domains, arrays, iterators, …
  - *task*: cobegins, atomic transactions, sync variables, …

- Composition of parallelism

- Global view of computation, data structures
Data Parallelism: Domains

- **domain**: an index set
  - specifies size and shape of arrays
  - supports sequential and parallel iteration
  - potentially decomposed across locales

- Three main classes:
  - **arithmetic**: indices are Cartesian tuples
    - rectilinear, multidimensional, optionally strided and/or sparse
  - **indefinite**: indices serve as hash keys
    - supports hash tables, associative arrays, dictionaries
  - **opaque**: indices are anonymous
    - supports sets, graph-based computations

- Chapel’s fundamental concept for data parallelism
Simple Domain Declarations

\[
\begin{align*}
\text{var } m & : \text{ int } = 4; \\
\text{var } n & : \text{ int } = 8; \\
\text{var } D : \text{ domain}(2) & = [1..m, 1..n];
\end{align*}
\]

\[ D \]
Simple Domain Declarations

\[
\begin{align*}
&\text{var } m: \text{ int } = 4; \\
&\text{var } n: \text{ int } = 8; \\
&\text{var } D: \text{ domain(2) } = [1..m, 1..n]; \\
&\text{var } D\text{Inner: subdomain}(D) = [2..m-1, 2..n-1];
\end{align*}
\]
Domain Uses

- **Declaring arrays:**
  \[
  \text{var } A, B: [D] \text{ float;}
  \]

- **Sub-array references:**
  \[
  A(DInner) = B(DInner);
  \]

- **Iteration:**
  \[
  \text{forall } (i,j) \text{ in } DInner \{ \ldots A(i,j) \ldots \}
  \]
  \[
  \text{or: forall } \text{ ind in } DInner \{ \ldots A(\text{ind}) \ldots \}
  \]
  \[
  \text{or: [ind in DInner] \ldots A(\text{ind})} \ldots
  \]

- **Array reallocation:**
  \[
  D = [1..2*m, 1..2*n];
  \]
Other Arithmetic Domains

\begin{verbatim}
var StridedD: subdomain(D) = D by (2,3);

var indexList: seq(index(D)) = ...;
var SparseD: sparse subdomain(D) = indexList;
\end{verbatim}
Task Parallelism

- **co-begins**: indicate statements that may run in parallel:
  ```
  computePivot(lo, hi, data);  
  cobegin  
  cobegin  
  {  
  ComputeTaskA(...);
  Quicksort(lo, pivot, data);
  Quicksort(pivot, hi, data);
  }  
  
  ComputeTaskB(...);
  }
  ```

- **atomic sections**: support atomic transactions
  ```
  atomic  
  
  {  
  newnode.next = insertpt;
  newnode.prev = insertpt.prev;
  insertpt.prev.next = newnode;
  insertpt.prev = newnode;
  }
  ```

- **sync and single-assignment variables**: synchronize tasks
  - similar to Cray MTA C/Fortran
Locality-aware Programming

- **locale**: architectural unit of storage and processing
- programmer specifies number of locales on executable command-line
  
  ```
prompt> myChapelProg -nl=8
  ```
- Chapel programs are provided with built-in locale array:
  ```
  const Locales: [1..numLocales] locale;
  ```
- Users may use it to create their own locale arrays:
  ```
  var CompGrid: [1..GridRows, 1..GridCols] locale = ...;
  ```
  
  ![CompGrid

  ```
  var TaskALocs: [1..numTaskALocs] locale = Locales(1..2);
  var TaskBLocs: [1..numTaskBLocs] locale = Locales(3..numLocales);
  ```
  
  ![TaskALocs ![TaskBLocs]
Data Distribution

- Domains may be distributed across locales
  
  ```chapel
  var D: domain(2) distributed(Block(2) on CompGrid) = ...;
  ```

- Distributions specify...
  - Mapping of indices to locales
  - Per-locale storage layout of domain indices and array elements

- Distributions implemented as a class hierarchy
  - Chapel provides a number of standard distributions
  - Users may also write their own
Computation Distribution

• “on” keyword binds computation to locale(s):
  \[
  \text{cobegin} \{
  \quad \text{on TaskALocs do ComputeTaskA}(...); \\
  \quad \text{on TaskBLocs do ComputeTaskB}(...);
  \}
  \]

• “on” can also be used in a data-driven manner:
  \[
  \text{forall (i,j) in D} \{
  \quad \text{on B(j/2,i*2) do A(i,j) = foo(B(j/2,i*2));}
  \}
  \]
Chapel Challenges

• User Acceptance
  • True of any new language
  • Skeptical audience

• Commodity Architecture Implementation
  • Chapel designed with idealized architecture in mind
  • Clusters are not ideal in many respects
  • Results in implementation and performance challenges

• And many others as well…
Outline

✓ Chapel Motivation & Foundations
✓ Chapel Overview

➢ Chapel Activities and Plans
Phase II Activities

- 2003-2006:
  - Application studies to drive language design
    - HPCC, NPB, SSCA benchmarks
    - kernels from Cray customer applications
    - other kernels of interest (connected components, FMM)
  - Design and specification of Chapel language
  - Implementation work on portable Chapel prototype
  - Outreach to inform users and get feedback
    - government: LANL, Sandia, LLNL, ORNL, JPL, NITRD
    - conferences: ICS, PPoPP, LCPC, PGAS, HIPS, HPL, LaR
    - mainstream industry: Microsoft (w/ AMD attendance)
    - HPCS: biannual reviews, SW productivity meetings
What’s next?

• **HPCS phase III**
  • July 2006 – December 2010
  • 2 vendors expected to be funded
  • proposals submitted May 5th

• **HPCS Language Effort forking off**
  • all 3 phase II language teams eligible for phase III
  • High Productivity Language Systems (HPLS) team
    • language experts/enthusiasts from national labs, academia
    • to study, evaluate the vendor languages, report to DARPA
    • July 2006 – December 2007

• DARPA hopes…
  …that a language consortium will emerge from this effort
  …to involve mainstream computing vendors as well
  …to avoid repeating mistakes of the past (Ada, HPF, …)
Proposed Phase III Activities

• **Short-term (2006-2007):**
  • support user evaluations of Chapel
    • HPCS mission partners
    • HPLS language evaluation team
    • software productivity team
    • other potential user communities
  • continue Chapel implementation
  • capture application studies as tutorials
  • revise language as suggested by these activities
• **Longer-term (2008-2010):**
  • participate in HPLS consortium language efforts
  • help build support for language in community
  • fold HPLS language into Cascade compiler, tools
Summary

• Chapel is being designed to…
  …enhance programmer productivity
  …address a wide range of HEC algorithms

• Via high-level, extensible abstractions for…
  …multithreaded parallel programming
  …locality-aware programming

• Status:
  • draft language specification available at:
    http://chapel.cs.washington.edu
  • Open source implementation proceeding apace
  • Your feedback desired!