

Chapel: Background

Sung-Eun Choi and Steve Deitz Cray Inc.

Chapel Settings



- HPCS: High Productivity Computing Systems (DARPA)
 - Goal: Raise HEC user productivity by 10x

 Productivity = Performance + Programmability + Portability + Robustness
- Phase II: Cray, IBM, Sun (July 2003 June 2006)
 - Evaluated entire system architecture
 - Three new languages (Chapel, X10, Fortess)
- Phase III: Cray, IBM (July 2006)
 - 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

Outline



- Chapel's Settings and Goals
- Chapel's Themes
 - Global-view abstractions
 - General parallel programming
 - Multiple levels of design
 - Control of locality
 - Mainstream language features





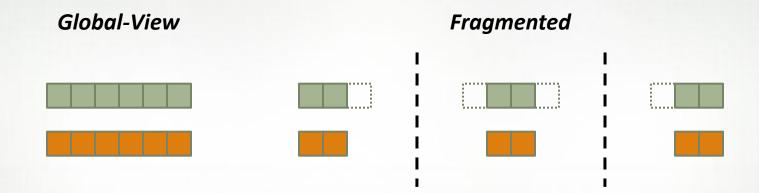
Definitions

- Programming model
 The mental model of a programmer
- Fragmented model
 Programmer takes point-of-view of a single processor/thread
- SPMD models (Single Program, Multiple Data)
 Fragmented models with multiple copies of one program
- Global-view model
 Programmer writes code to describe computation as a whole





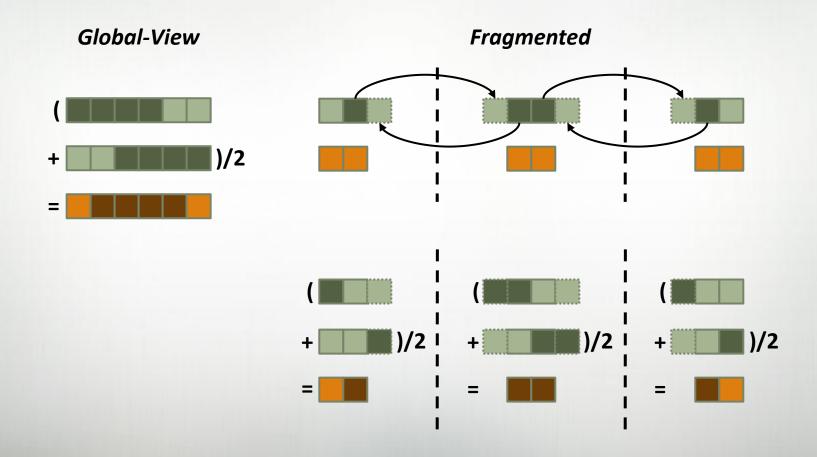
Example: 3-Point Stencil (Data Declarations)







Example: 3-Point Stencil (Computation)







Example: 3-Point Stencil (Code)

Global-View

```
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;
}
```

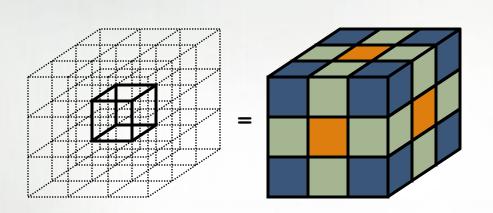
Assumes p divides n

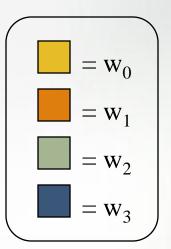
Fragmented

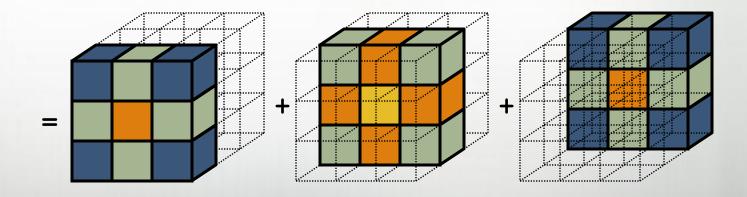
```
def main() {
  var n = 1000;
  var me = commRank(), p = commSize(),
    myN = n/p, myLo = 1, myHi = myN;
  var A, B: [0..myN+1] real;
  if me < p {
    send (me+1, A(myN));
    recv (me+1, A (myN+1));
  } else myHi = myN-1;
  if me > 1 {
    send (me-1, A(1));
    recv (me-1, A(0));
  } else myLo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

NAS MG Stencil













```
use caf_intrinsics
implicit none
include 'cafnpb.h
integer n1 n2 n3 kk
 double precision u(n1,n2,n3)
integer axis
if( .not. dead(kk) )then
    do axis = 1, 3
if(nprocs .ne. 1) then
    call sync all()
call give3( axis, +1, u, n1,
n2, n3, kk)
    call give3( axis, -1, u, n1, n2, n3, kk)
          call sync all()
    call take3( axis, +1, u, n1, n2, n3)
       else
    call commlp(axis, u, n1, n2, n3, kk)
   endif
enddo
      call sync_all()
call sync_all()
    call zero3(u,n1,n2,n3)
return
 implicit none
 include 'globals.h' subroutine
comm3(u,n1,n2,n3,kk)
integer axis, dir, n1, n2, n3, k, ierr
double precision u ( n1, n2, n3 )
integer i3, i2, i1, buff len,buff id
buff len = 0
if( axis .eq. 1 ) then
    if( dir .eq. -1 ) then
       do i3=2.n3-1
             buff len = buff len + 1
     buff(buff_len,buff_id) = u(2, i2,i3)
          enddo
       enddo
    buff(1:buff_len,buff_id+1)[nbr(axis
,dir,k)] =
      buff(1:buff len,buff id)
    else if( dir .eq. +1 ) then
          do i2=2,n2-1
    buff_len = buff_len + 1
buff(buff_len, buff_id) =
u(n1-1, i2,i3)
       enddo
enddo
    buff(1:buff_len,buff_id+1)[nbr(axis
,dir,k)] =
       buff(1:buff_len,buff_id)
 if( axis .eq. 2 ) then
    if (dir .eg. -1 )then
```

```
do i3=2,n3-1
              do i1=1,n1
                  buff len = buff len + 1
         buff(buff_len, buff_id) = u(i1, 2,i3)
         buff(1:buff_len,buff_id+1)[nbr(axis
         else if( dir .eq. +1 ) then
           do i3=2,n3-1
              do i1=1,n1
                  buff len = buff len + 1
         buff(buff_len, buff_id)=
u(i1,n2-1,i3)
         buff(1:buff len.buff id+1)[nbr(axis
           buff(1:buff len.buff id)
        endi f
    if( axis .eq. 3 ) then if( dir .eq. -1 ) then
           do i2=1.n2
               do i1=1.n1
         buff_len = buff_len + 1
buff(buff_len, buff_id) =
u(i1,i2,2)
              enddo
           enddo
         buff(1:buff_len,buff_id+1)[nbr(axis
           buff(1:buff_len,buff_id)
        else if (dir .eg. +1 ) then
           do i2=1,n2
               do i1=1,n1
         buff_len = buff_len + 1
buff(buff_len, buff_id) =
u(i1,i2,n3-1)
         buff(1:buff_len,buff_id+1)[nbr(axis
,dir,k)] =
           buff(1:buff_len,buff_id)
    endif
     subroutine take3 (axis, dir, u, n1, n2,
     use caf_intrinsics
     implicit none
     include 'globals.h'
    integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )
     integer buff id, indx
     integer i3, i2, i1
     if (axis .eq. 1 ) then
           do i3=2.n3-1
                  indx = indx + 1
```

```
u(n1,i2,i3) = buff(indx,buff id)
      enddo
enddo
   else if( dir .eq. +1 ) then
          do i2=2,n2-1
    u(1,i2,i3) = buff(indx,
buff_id)
       enddo
enddo
   endif
   if ( dir .eq. -1 ) then
          do i1=1,n1
indx = indx + 1
              u(i1,n2,i3) = buff(indx)
       enddo
enddo
   else if( dir .eq. +1 ) then
       do i3=2.n3-1
    u(i1,1,i3) = buff(indx,
buff id)
              indx = indx + 1
       enddo
   endif
if( axis .eq. 3 ) then if( dir .eq. -1 ) then
      do i2=1,n2
do i1=1,n1
     \begin{array}{ccc} & \max = & \text{indx} + 1 \\ & \text{u(i1,i2,n3)} & = & \text{buff(indx)} \\ \text{buff id)} \end{array} 
       enddo
enddo
   else if( dir .eq. +1 ) then
          do i1=1,n1
              u(i1,i2,1) = buff(indx)
    buff id
       enddo
endif
return
subroutine commlp( axis, u, n1, n2, n3, kk)
use caf_intrinsics
integer axis, dir, n1, n2, n3
double precision u( n1, n2, n3 )
integer i3, i2, i1, buff len,buff id
integer i, kk, indx
buff id = 3 + dir
buff_len = nm2
   buff(i,buff id) = 0.0D0
```

```
enddo
      buff_id = 3 + dir
      buff len = nm2
         buff(i,buff_id) = 0.0D0
      buff id = 2 + dir
      if( axis .eq. 1 )then
do i3=2,n3-1
do i2=2,n2-1
          buff_len = buff_len + 1
buff(buff_len, buff_id) = u(
n1-1, i2,i3)
             enddo
      endif
      if(axis .eq. 2)then
do i3=2,n3-1
             do i1=1,n1
               buff len = buff len + 1
                 buff(buff_len, buff_id )= u(
          i1,n2-1,i3)
             enddo
       endif
if( axis .eq. 3 )then
do i2=1,n2
            do i1=1,n1
                buff_len = buff_len + 1
          buff(buff_len, buff_id) = u(
i1,i2,n3-1)
             enddo
      buff len = 0
      if( axis .eq. 1 )then
do i3=2,n3-1
             do i2=2,n2-1
                buff_len = buff_len + 1
          buff(buff_len,buff_id) = u(
2, i2,i3)
         enddo
      if( axis .eq. 2 )then
do i3=2,n3-1
             do i1=1,n1
                buff len = buff len + 1
          buff(buff_len, buff_id) = u(
i1, 2,i3)
      if( axis .eq. 3 ) then
         do i2=1.n2
             do i1=1,n1
                buff len = buff len + 1
                 buff(buff len, buff id ) = u(
          11.12.21
             enddo
      do i=1,nm2
buff(i,4) = buff(i,3)
         buff(i,2) = buff(i,1)
```

```
do i2=2,n2-1
               indx = indx + 1
          u(n1,i2,i3) = buff(indx,
buff_id)
     if(axis .eq. 2)then
do i3=2,n3-1
            do i1=1,n1
               indx = indx + 1
         u(i1,n2,i3) = buff(indx.
buff id)
        enddo
enddo
     if(axis .eq. 3)then
        do i2=1,n2
do i1=1,n1
               indy = indy + 1
         u(i1,i2,n3) = buff(indx,
buff id)
        enddo
enddo
     buff_id = 3 + dir
indx = 0
     if( axis .eq. 1 ) then
do i3=2,n3-1
           do i2=2,n2-1
indx = indx
          u(1,i2,i3) = buff(indx,
buff id)
     endif
if(axis .eq. 2)then
        do i3=2,n3-1
                indv = indv + 1
                u(i1,1,i3) = buff(indx,
          buff id )
      endif
     if(axis .eq. 3)then
do i2=1,n2
            do i1=1,n1
               indx = indx + 1
         u(i1,i2,1) = buff(indx, buff id)
            enddo
      endif
      return
     end
     subroutine
rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k
      implicit none
     include 'cafnph.h
     integer m1k, m2k, m3k, m1j, m2j, m3j,k
     double precision r(m1k,m2k,m3k)
s(m1j,m2j,m3j)
     integer j3, j2, j1, i3, i2, i1, d1, d2,
d3, j
     double precision x1(m), y1(m), x2,y2
     if (mlk.eq.3) then
       d1 = 2
```

d1 = 1

endif

do i3=2,n3-1

```
if (m2k.eq.3) then
  d2 = 2
else
  d2 = 1
endif
if (m3k.eq.3) then
  d3 = 2
else
andif.
  i3 = 2*j3-d3
  do 12=2.m21-1
    do j1=2,m1j
       i1 = 2*i1-d1
    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)
    y1(i1-1) = r(i1-1,i2-1,i3-1) + r(i1-1,i2-1,i3+1)
    r(i1-1,i2+1,i3-1) + r(i1-1,i2+1,i3-1) +
    enddo
     do j1=2,m1j-1
    i1 = 2*j1-d1
y2 = r(i1, i2-1,i3-1) + r(i1,
i2-1,i3+1)
    + r(i1, i2+1,i3-1) + r(i1, i2+1,i3+1)
    x2 = r(i1, i2-1,i3 ) + r(i1, i2+1,i3 )
    + r(i1, i2, i3-1) + r(i1, i2, i3+1)
       s(j1,j2,j3) = 0.500 * r(i1,i2,i3)
    + 0.25D0 * (r(i1-1,i2,i3) +
r(i1+1,i2,i3) + x2)
        + 0.125D0 * ( x1(i1-1) +
        + 0.0625D0 * ( y1(i1-1) +
    v1(i1+1))
     enddo
  enddo
  j = k-1
   call comm3(s,m1j,m2j,m3j,j)
  return
  end
```

if(axis .eg. 1) then



NAS MG Stencil in Chapel

Our previous work in ZPL has shown that such compact codes can result in better performance than the Fortran + MPI.



Summary of Current Programming Systems

	System	Data Model	Compute Model
Communication Libraries	MPI/MPI-2	Fragmented	Fragmented
	SHMEM	Fragmented	Fragmented
	ARMCI	Fragmented	Fragmented
	GASNet	Fragmented	Fragmented
Shared Memory	OpenMP, pThreads	Global-View (trivially)	Global-View (trivially)
PGAS Languages	Co-Array Fortran	Fragmented	Fragmented
	UPC	Global-View	Fragmented
	Titanium	Fragmented	Fragmented
HPCS Languages	Chapel	Global-View	Global-View
	X10 (IBM)	Global-View	Global-View
	Fortress (Sun)	Global-View	Global-View

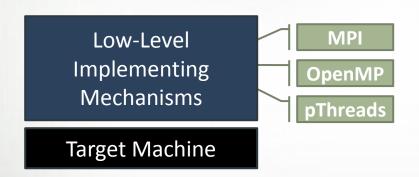


General Parallel Programming

- Express all parallelism in the software
 - Forms: data, task, nested (arbitrary composition thereof)
 - Levels: module, function, loop, statement
- Target all parallelism in the hardware
 - Systems: multicore desktops, clusters, HPC systems
 - Types: multithreading, vector
 - Levels: across cores, across nodes, across systems







"Why is everything so difficult?"



Target Machine

"Why can't I optimize this?"

Multiple Levels of Design



Task Scheduling

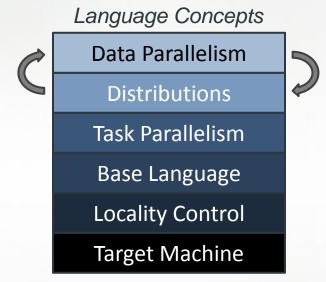
Work Stealing

Suspendable Tasks

Task Pool

Thread per Task

Target Machine



Memory Management

Garbage Collection

Region-Based

Per-Allocation Frees

Target Machine





Given

- Scalable systems tend to store memory with processors
- Remote accesses tend to take longer than local accesses

Therefore

- Placement of data relative to computation matters
- Programmers need control over data placement

Note

- As multi-core chips grow, locality may matter on desktops
- GPUs/accelerators expose node-level locality



Mainstream Language Features

- Object-oriented programming with value and reference classes
- Generic programming with types and compile-time constants
- Latent typing and a rich set of primitive types
- Modules for libraries and code organization
- Functions with nesting, overloading, and named arguments
- Multi-dimensional and associative arrays with slicing, etc.
- Classes, records, and unions
- Tuples, ranges, and domains
- Standard modules (e.g., Math, Random, Time, BitOps, Norm)

Questions?



- Chapel's Settings and Goals
- Chapel's Design
 - Global-view abstractions
 - General parallel programming
 - Multiple levels of design
 - Control of locality
 - Mainstream language features