Five Powerful Chapel Idioms

Steve Deitz

Cray Inc.

What is Chapel?

- A new parallel language
  - Under development at Cray Inc.
  - Supported through the DARPA HPCS program
  - Abstractions from ZPL, HPF, Cray XMT C, ...
- With many powerful idioms, features, and functions
  - Asynchronous and synchronous remote tasks
  - Data parallelism when applicable
  - User-defined distributions
  - Local and remote transactions
  - Arbitrarily nested parallelism
  - ...

Chapel Productivity Goals

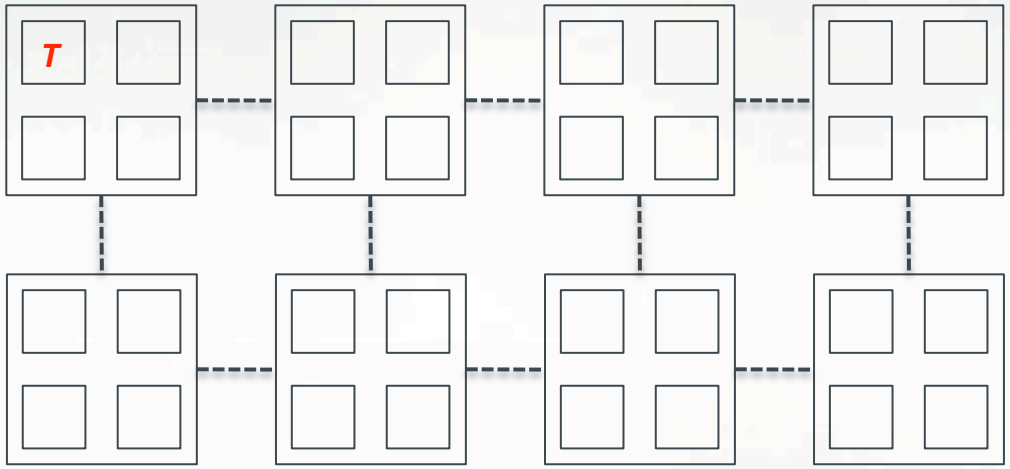
- Improve **programmability** over current languages
  - Writing parallel codes
  - Reading, changing, porting, tuning, maintaining, ...
- Support **performance** at least as good as MPI
  - Competitive with MPI on generic clusters
  - Better than MPI on more capable architectures
- Improve **portability** 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

- General parallel programming
  - Express all levels of software parallelism
  - Target all levels of hardware parallelism
- Partitioned Global Address Space (PGAS)
- Global-view abstractions
- Multiple levels of design
- Control of locality
- Mainstream language features
  - From scripting languages for fast prototyping
  - From object-oriented languages for robust designs

Chapel Parallel Programming Model

- Single task executes main() on Locale 0



- Advantages over SPMD
  - Single (global) flow of control
  - Fragmentation of problem is unnecessary (though possible)

CRAY
THE SUPERCOMPUTER COMPANY

Easy Invocation of Remote Asynchronous and Synchronous Tasks

Easy Invocation of Remote Tasks

- Syntax

```
on-statement:
  on expression statement
```

- Semantics
  - Evaluates expression to determine locale
  - Executes statement on locale
- Example

```
on object { update(object); }

on A(i) { A(i) = B(i) + f(i); }
```

Easy Invocation of Asynchronous Tasks

- Syntax

```
begin-statement:
  begin statement
```

- Semantics
  - Executes statement in a concurrent task
  - Control continues immediately to next statement
- Example

```
sync {
  begin f1();
  f2();
}
```

Easy Invocation of Remote Asynchronous Tasks

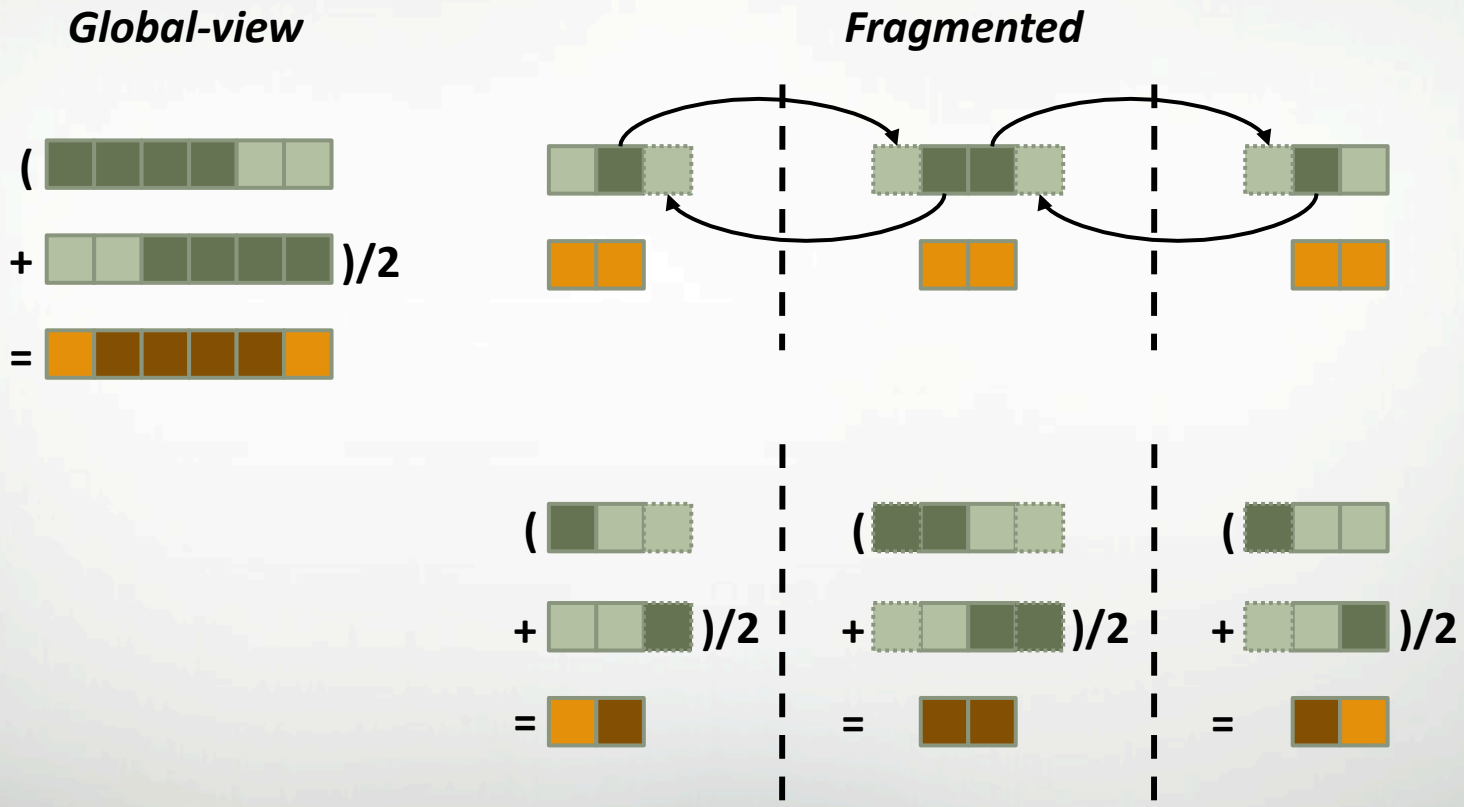
```
begin on A(i) {
  A(i) += f(i);
}
```

CRAY
THE SUPERCOMPUTER COMPANY

Simple Support for Data Parallelism When it is Applicable

Global-View vs. Fragmented Computation

Contrasted depictions of a 3-point stencil



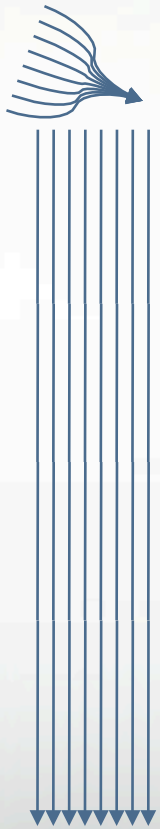
Global-View vs. Fragmented Code

Contrasted codes of a 3-point stencil

Global-view

```
def main() {
  var n = 1000;
  const D: domain(1) = [1..n];
  var A, B: [D] real;

  forall i in 2..n-1 do
    B(i) = (A(i-1)+A(i+1))/2;
}
```



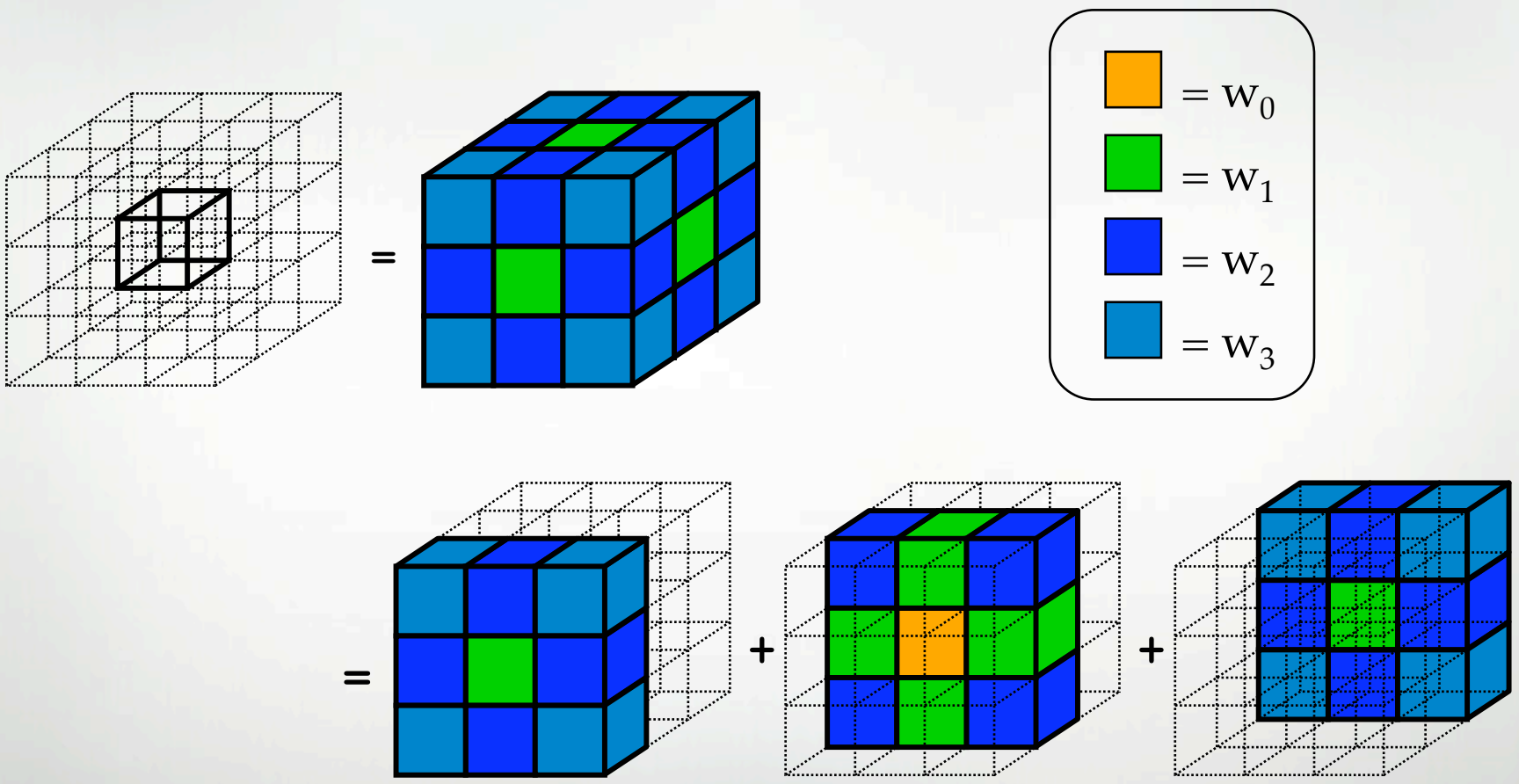
Fragmented

```
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;
}
```

Assumes p divides n

NAS MG Stencil



NAS MG Stencil in Fortran + MPI

```
use caf_intrinsics

 implicit none

 include 'cafnpb.h'
 include 'globals.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 )
          call take3( axis, +1, u, n1,
     n2, n3 )
       else
          call comm1p( axis, u, n1, n2,
     n3, kk )
       endif
    enddo
 else
    do  axis = 1, 3
       call sync_all()
       call sync_all()
    enddo
    call zero3(u,n1,n2,n3)
 endif
 return
 end

 subroutine give3( axis, dir, u, n1, n2,
     n3, k )
 use caf_intrinsics


 implicit none

 include 'cafnpb.h'
 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_id = 2 + dir
 buff_len = 0

 if( axis .eq.  1 )then
    if( dir .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
       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  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
       enddo

       buff(1:buff_len,buff_id+1)
      [nbr(axis,dir,k)] =
>       buff(1:buff_len,buff_id)

    endif
 endif

 if( axis .eq.  2 )then
    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,  2,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  i3=2,n3-1
          do  i1=1,n1
             buff_len = buff_len + 1
             buff(buff_len,  buff_id )=
     u( i1,n2-1,i3)
          enddo
       enddo

       buff(1:buff_len,buff_id+1)
      [nbr(axis,dir,k)] =
>       buff(1:buff_len,buff_id)

    endif
 endif

 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,dir,k)] =
>       buff(1:buff_len,buff_id)

    else 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,n3-1)
          enddo
       enddo

       buff(1:buff_len,buff_id+1)
      [nbr(axis,dir,k)] =
>       buff(1:buff_len,buff_id)

    endif
 endif

 return
 end


 subroutine take3( axis, dir, u, n1, n2,
     n3 )
 use caf_intrinsics


 implicit none

 include 'cafnpb.h'
 include 'globals.h'

 integer axis, dir, n1, n2, n3
 double precision u( n1, n2, n3 )

 integer buff_id, indx

 integer i3, i2, i1

 buff_id = 3 + dir
 indx = 0

 if( axis .eq.  1 )then
    if( dir .eq. -1 )then

       do  i3=2,n3-1
          do  i2=2,n2-1
             indx = indx + 1
             u(n1,i2,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    else if( dir .eq. +1 ) then

       do  i3=2,n3-1
          do  i2=2,n2-1
             indx = indx + 1
             u(1,i2,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    endif
 endif

 if( axis .eq.  2 )then
    if( dir .eq. -1 )then

       do  i3=2,n3-1
          do  i2=2,n2-1
             indx = indx + 1
             u(i1,n2,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    else if( dir .eq. +1 ) then

       do  i3=2,n3-1
          do  i1=1,n1
             indx = indx + 1
             u(i1,1,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    endif
 endif

 if( axis .eq.  3 )then
    if( dir .eq. -1 )then

       do  i2=1,n2
          do  i1=1,n1
             indx = indx + 1
             u(i1,i2,n3) = buff(indx,
     buff_id )
          enddo
       enddo

    else if( dir .eq. +1 ) then

       do  i2=1,n2
          do  i1=1,n1
             indx = indx + 1
             u(i1,i2,1) = buff(indx,
     buff_id )
          enddo
       enddo

    endif
 endif

 return
 end


 subroutine comm1p( axis, u, n1, n2, n3,
     kk )
 use caf_intrinsics


 implicit none

 include 'cafnpb.h'
 include 'globals.h'

 integer axis, dir, n1, n2, n3
 double precision u( n1, n2, n3 )

 integer i3, i2, i1, buff_len,buff_id
 integer i, kk, indx

 dir = -1

 buff_id = 3 + dir
 buff_len = nm2

 do  i=1,nm2
    buff(i,buff_id) = 0.0D0
 enddo


 dir = +1

 buff_id = 3 + dir
 buff_len = nm2

 do  i=1,nm2
    buff(i,buff_id) = 0.0D0
 enddo

 dir = +1

 buff_id = 2 + dir
 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( n1-1, 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  i3=2,n3-1
       do  i2=2,n2-1
          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)

 endif

 if( axis .eq.  2 )then
    if( dir .eq. -1 )then

       do  i3=2,n3-1
          do  i2=2,n2-1
             indx = indx + 1
             u(n1,i2,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    else if( dir .eq. +1 ) then

       do  i3=2,n3-1
          do  i1=1,n1
             indx = indx + 1
             u(i1,n2,i3) = buff(indx,
     buff_id )
          enddo
       enddo

    endif
 endif

 if( axis .eq.  3 )then
    if( dir .eq. -1 )then

       do  i2=1,n2
          do  i1=1,n1
             indx = indx + 1
             u(i1,i2,n3) = buff(indx,
     buff_id )
          enddo
       enddo

    else if( dir .eq. +1 ) then

       do  i2=1,n2
          do  i1=1,n1
             indx = indx + 1
             u(i1,i2,1) = buff(indx,
     buff_id )
          enddo
       enddo

    endif
 endif

 return
 end


 subroutine
     rprj3(r,m1k,m2k,m3k,s,m1j,m2j,m3j,k
     )
 implicit none
 include 'cafnpb.h'
 include 'globals.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(m1k.eq.3)then
   d1 = 2
 else
   d1 = 1
 endif

 if(m2k.eq.3)then
   d2 = 2
 else
   d2 = 1
 endif

 if(m3k.eq.3)then
   d3 = 2
 else
   d3 = 1
 endif

 do  j3=2,m3j-1
   i3 = 2*j3-d3
   do  j2=2,m2j-1
     i2 = 2*j2-d2
     do j1=2,m1j
       i1 = 2*j1-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) =
>        0.5D0 * r(i1,i2,i3)
>      + 0.25D0 * (r(i1-1,i2,i3) +
     r(i1+1,i2,i3) + x2)
>      + 0.125D0 * ( x1(i1-1) +
     x1(i1+1) + y2)
>      + 0.0625D0 * ( y1(i1-1) +
     y1(i1+1) )
     enddo
   enddo
 enddo
 j = k-1
 call comm3(s,m1j,m2j,m3j,j)
 return
 end
```

NAS MG Stencil in Chapel

```
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:

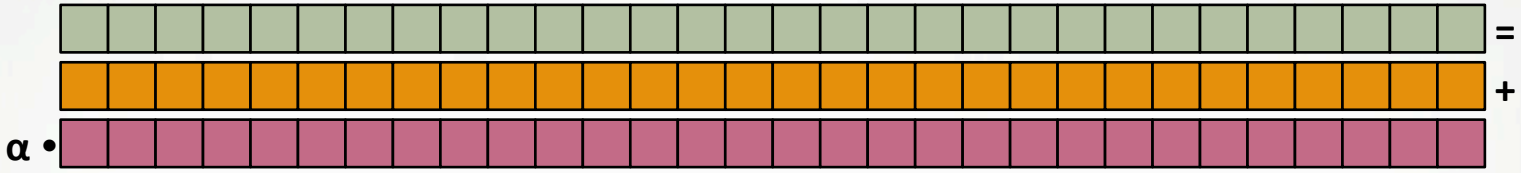
B. L. Chamberlain, S. J. Deitz, and L. Snyder. *A comparative study of the NAS MG benchmark across parallel languages and architectures*. In Proceedings of the ACM Conference on Supercomputing, 2000.

CRAY
THE SUPERCOMPUTER COMPANY
Support for User-Defined Distributions

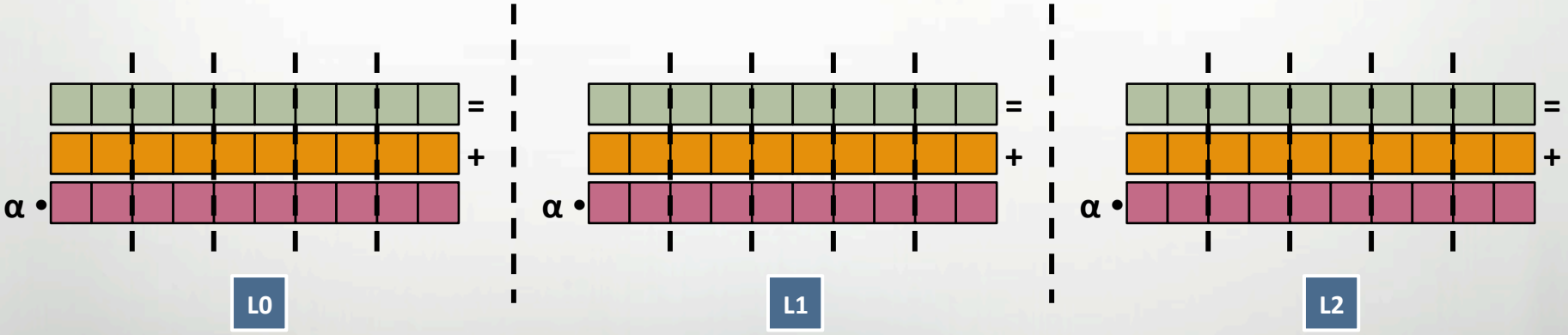
What is a Distribution?

A "recipe" for distributed arrays that...

Instructs the compiler how to Map the global view...



...to a fragmented, per-processor implementation

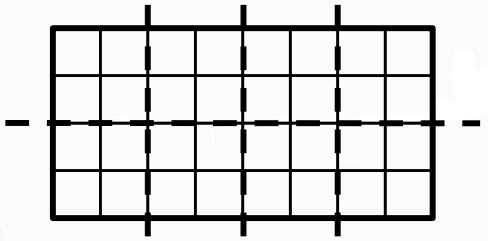


Distributing a Domain

Domains are associated to a distribution

```
const Dist = new Block(rank=2, bbox=[1..4, 1..8]);

var Dom: domain(2) distributed Dist = [1..4, 1..8];
```



distributed over

The distribution defines:

- Ownership of domain indices and array elements
- Default distribution of work (task-to-locale map)

  E.g., forall loops over distributed domains/arrays

Authoring Distributions

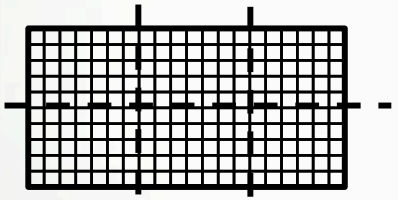
- (Advanced) programmers can write distributions
- Built-in library of distributions
  - No extra compiler support for built-in distributions
  - Compiler uses structural interface:
    - Create domains and arrays
    - Map indices to locales
    - Access array elements
    - Iterate over indices/elements sequentially, in parallel, zippered
    - …
- Distributions are built using language-level concepts
  - On for data and task locality
  - Begin, cobegin, and coforall for data parallelism

Distributing Domains

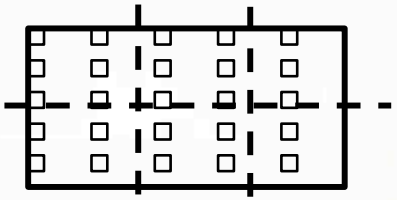
All domain types can be distributed.

Semantics are independent of distribution.

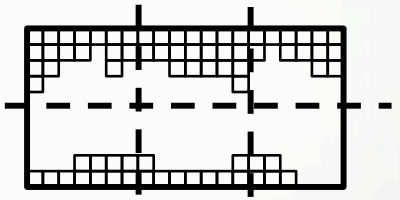
(Though performance and parallelism will vary...)



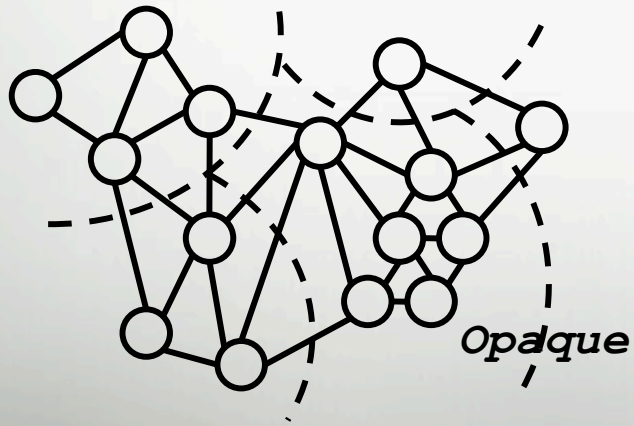
Dense



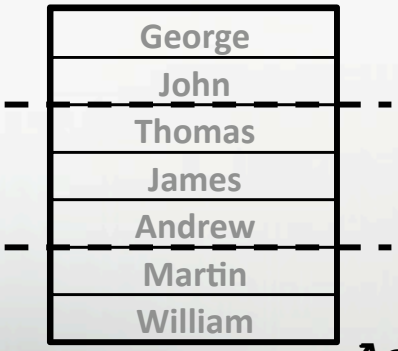
Strided



Sparse



Opaque



Associative

Exposing GPUs via User-Defined Distributions

2009 Summer Internship: Albert Sidelnik from UIUC

- Added a distribution that maps data to GPUs
- Changed distribution of domain for HPCC STREAM
- Minor compiler changes (to generate CUDA, etc.)

```
const Dist = new GPUDist(rank=1, tpb=256);

const Dom: domain(1) distributed Dist = [1..m];

var A, B, C: [Dom] real;

forall (a,b,c) in (A,B,C) do
  a = b + alpha * c;
```

CRAY
THE SUPERCOMPUTER COMPANY

Support for Local and Remote Transactions

Atomic Transactions (Work in Progress)

- Syntax

```
atomic-statement:
  atomic statement
```

- Semantics
  - Executes statement as if it is a single operation
  - No other task sees a partial result
- Example

```
atomic A(i) = A(i) + 1;
```

```
atomic {
  newNode.next = node;
  newNode.prev = node.prev;
  node.prev.next = newNode;
  node.prev = newNode;
}
```

CRAY
THE SUPERCOMPUTER COMPANY

Support for Arbitrarily Nested Parallelism

Nested Data Parallel Tasks

Example of invoking two data-parallel tasks

```
sync {
  begin A = B + alpha * C;
  begin D = E + alpha * F;
}
```

More Chapel...

- Full day tutorial
  - Upcoming joint tutorial with X10 and UPC at SC '09
- Download the release
  - http://sourceforge.net/projects/chapel/
- Contact us
  - Send us mail at chapel_info@cray.com
  - Visit our web page at http://chapel.cray.com/
  - View archives of chapel-users@lists.sourceforge.net
- Position paper
  - http://chapel.cray.com/LACSS09_DEITZ.pdf