Chapel: Background

Chapel's Origins

- **HPCS**: High Productivity Computing Systems
  - Overall goal: Raise high-end user productivity by 10x

    Productivity = Performance + Programmability + Portability + Robustness
- **Phase II:** Cray, IBM, Sun (July 2003 – June 2006)
  - Goal: Propose new productive system architectures
  - Each vendor created a new programming language
    - **Cray:** Chapel
    - **IBM:** X10
    - **Sun:** Fortress
- **Phase III:** Cray, IBM (July 2006 –)
  - Goal: Develop the systems proposed in phase II
  - Each vendor implemented a compiler for their language
    - Sun also continued their Fortress effort without HPCS funding

Chapel's Productivity Goals

- Vastly improve **programmability** over current languages
  - Writing parallel programs
  - 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** over current languages
  - As ubiquitous as MPI but more abstract
  - More portable than OpenMP, UPC, and CAF are thought to be
- Improve **robustness** via improved semantics
  - Eliminate common error cases
  - Provide better abstractions to help avoid other errors

Outline

- Chapel's Context
- Chapel's Motivating Themes
  1. General parallel programming
  2. *Global-view* abstractions
  3. *Multiresolution* design
  4. Control over locality/affinity
  5. Reduce gap between mainstream & HPC languages

1) General Parallel Programming

With a unified set of concepts...

...express any parallelism desired in a user's program

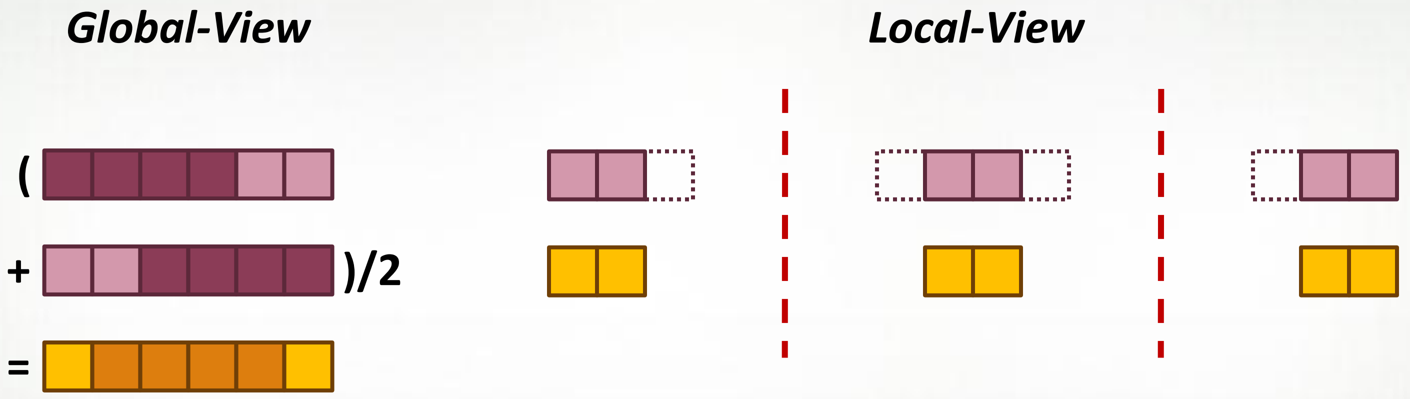
- **Styles:** data-parallel, task-parallel, concurrency, nested, …
- **Levels:** model, function, loop, statement, expression

...target all parallelism available in the hardware

- **Systems:** multicore desktops, clusters, HPC systems, …
- **Levels:** machines, nodes, cores, instructions

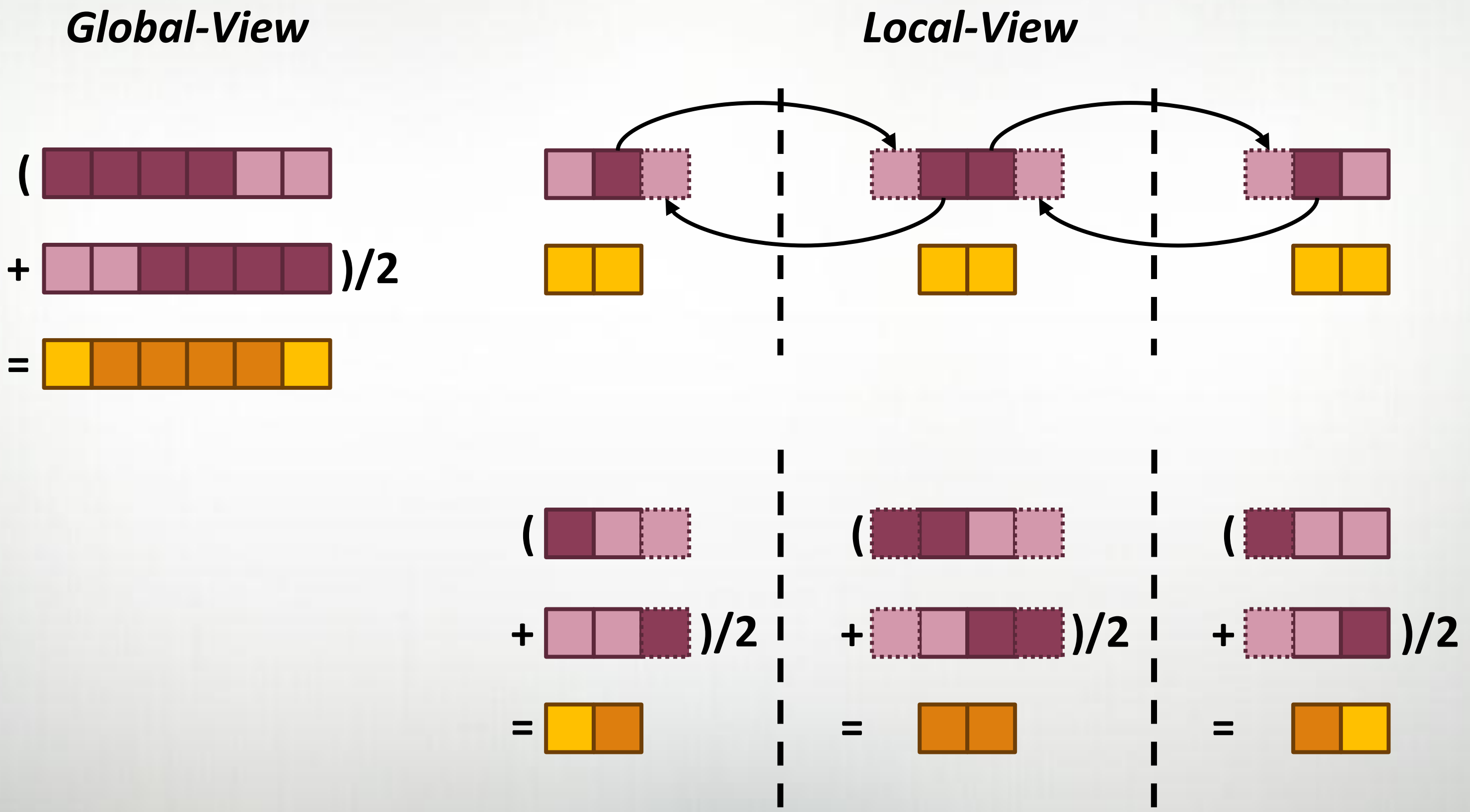
2) Global-View Abstractions

In pictures: "Apply a 3-Point Stencil to a vector"



2) Global-View Abstractions

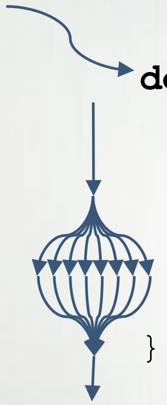
In pictures: "Apply a 3-Point Stencil to a vector"



2) Global-View Abstractions

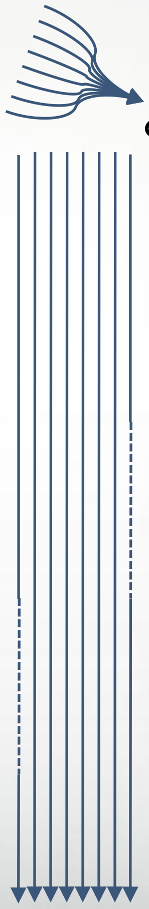
In code: "Apply a 3-Point Stencil to a vector"

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;
}
```

Local-View (SPMD)

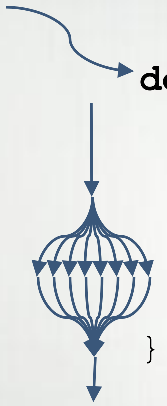


```
def main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  }
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  }
  forall i in 1..myN do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

Bug: Refers to uninitialized values at ends of A

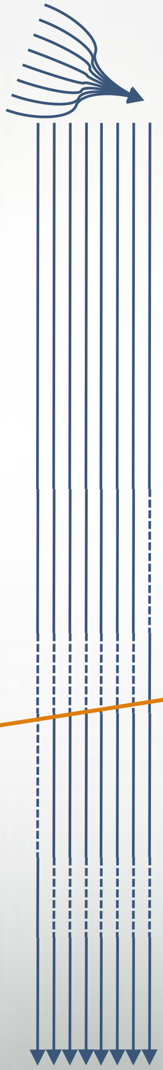
2) Global-View Abstractions

In code: “Apply a 3-Point Stencil to a vector”



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;
}
```



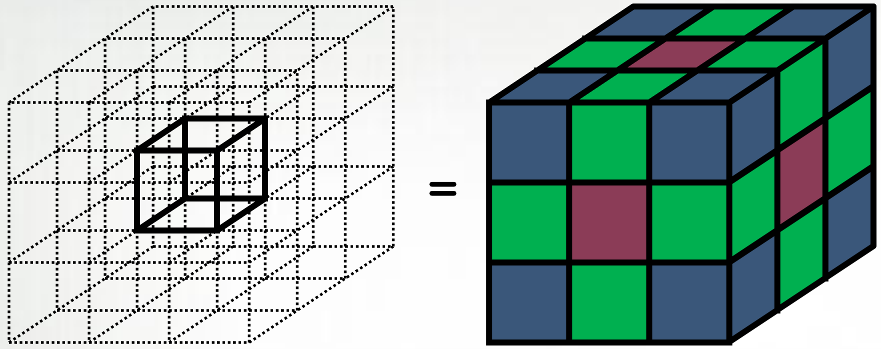
Local-View (SPMD)

```
def main() {
  var n = 1000;
  var p = numProcs(),
      me = myProc(),
      myN = n/p,
      iLo = 1,
      iHi = myN;
  var A, B: [0..myN+1] real;

  if (me < p-1) {
    send(me+1, A[myN]);
    recv(me+1, A[myN+1]);
  } else
    myHi = myN-1;
  if (me > 0) {
    send(me-1, A[1]);
    recv(me-1, A[0]);
  } else
    myLo = 2;
  forall i in iLo..iHi do
    B[i] = (A[i-1] + A[i+1])/2;
}
```

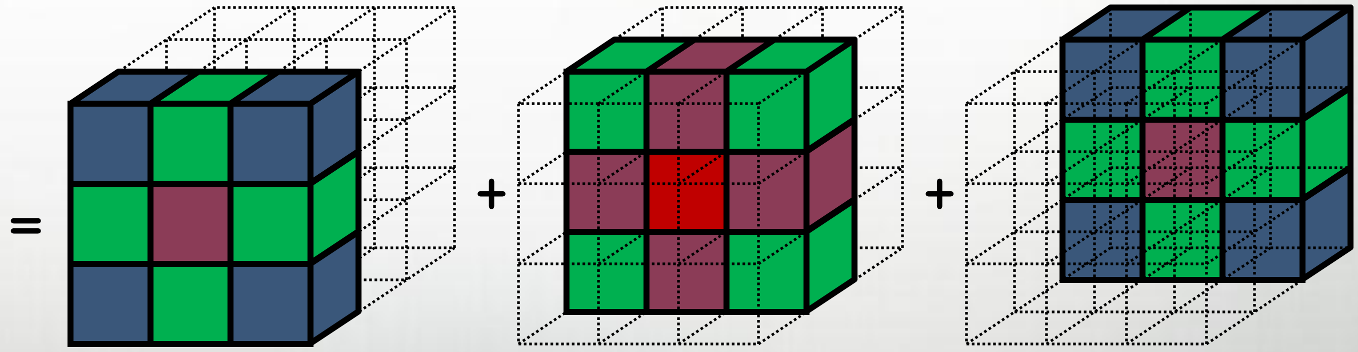
Assumes p divides n

Communication becomes geometrically more complex for higher-dimensional arrays

2) *rprj3* Stencil from NAS MG



🟥	= w_0
🟪	= w_1
🟩	= w_2
🟦	= w_3

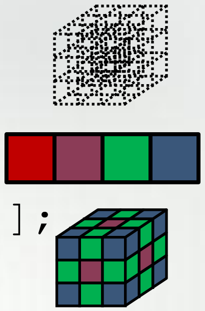


2) *rprj3* Stencil from NAS MG in Fortran + MPI

```fortran
      subroutine rprj3( r,m1k,m2k,m3k,s,m1j,m2j,m3j,k )

      implicit none

      include 'mpinpb.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)

      if( debug_vec(0) .ge. 1 )then
         call rep_nrm(s,m1j,m2j,m3j,'   rprj3',k-1)
      endif

      if( debug_vec(4) .ge. k )then
         call showall(s,m1j,m2j,m3j)
      endif

      return
      end

      subroutine norm2u3(r,n1,n2,n3,rnm2,rnmu,nx,ny,nz)

      implicit none

      include 'mpinpb.h'

      integer n1, n2, n3, nx, ny, nz
      double precision rnm2, rnmu, r(n1,n2,n3)
      double precision s, a, ss
      integer i3, i2, i1, ierr

      double precision dn

      dn = 1.0d0*nx*ny*nz

      s=0.0D0
      rnmu = 0.0D0
      do  i3=2,n3-1
         do  i2=2,n2-1
            do  i1=2,n1-1
               s=s+r(i1,i2,i3)**2
               a=abs(r(i1,i2,i3))
               if(a.gt.rnmu)rnmu=a
            enddo
         enddo
      enddo

      call mpi_allreduce(rnmu,ss,1,dp_type,
     >     mpi_max,mpi_comm_world,ierr)
      rnmu = ss
      call mpi_allreduce(s, ss, 1, dp_type,
     >     mpi_sum,mpi_comm_world,ierr)
      s = ss
      rnm2=sqrt( s / dn )

      return
      end

      subroutine rep_nrm(u,n1,n2,n3,title,kk)

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

      integer n1, n2, n3, kk
      double precision u(n1,n2,n3)
      character*8 title
      double precision rnm2, rnmu

      call norm2u3(u,n1,n2,n3,rnm2,rnmu,nx(kk),ny(kk),nz(kk))
      if( me .eq. root )then
         write(*,7)kk,title,rnm2,rnmu
 7       format(' Level',i2,' in ',a8,': norms =',D21.14,D21.14)
      endif

      return
      end

      subroutine comm3(u,n1,n2,n3,kk)

      implicit none

      include 'mpinpb.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 ready( axis, -1, kk )
               call ready( axis, +1, kk )

               call give3( axis, +1, u, n1, n2, n3, kk )
               call give3( axis, -1, u, n1, n2, n3, kk )

               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
         call zero3(u,n1,n2,n3)
      endif
      return
      end

      subroutine ready( axis, dir, k )

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

      integer axis, dir, k
      integer buff_id,buff_len,i,ierr

      buff_id = 3 + dir
      buff_len = nm2

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

      msg_id(axis,dir,1) = msg_type(axis,dir) +1000*me

      call mpi_irecv( buff(1,buff_id), buff_len,
     >     dp_type, nbr(axis,-dir,k), msg_type(axis,dir),
     >     mpi_comm_world, msg_id(axis,dir,1), ierr)
      return
      end

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

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

      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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         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

            call mpi_send(
     >           buff(1, buff_id ), buff_len,dp_type,
     >           nbr( axis, dir, k ), msg_type(axis,dir),
     >           mpi_comm_world, ierr)

         endif
      endif

      return
      end

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

      implicit none

      include 'mpinpb.h'
      include 'globals.h'

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

      integer buff_id, indx

      integer status(mpi_status_size), ierr

      integer i3, i2, i1

      call mpi_wait( msg_id( axis, dir, 1 ),status,ierr)
      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  i1=1,n1
                  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 )

      implicit none

      include 'mpinpb.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
      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
         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
         enddo
      endif

      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( 2,  i2,i3)
            enddo
         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,  2,i3)
            enddo
         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,i2,2)
            enddo
         enddo
      endif

      do  i=1,nm2
         buff(i,4) = buff(i,3)
         buff(i,2) = buff(i,1)
      enddo

      dir = -1

      buff_id = 3 + dir
      indx = 0

      if( axis .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
      endif

      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
      endif

      if( axis .eq.  3 )then
         do  i2=1,n2
            do  i1=1,n1
               indx = indx + 1
               u(i1,i2,n3) = buff(indx, buff_id )
            enddo
         enddo
      endif

      dir = +1

      buff_id = 3 + dir
      indx = 0

      if( axis .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

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

      return
      end

      subroutine showall(z,n1,n2,n3)

      implicit none

      include 'mpinpb.h'

      integer n1,n2,n3,i1,i2,i3,i,ierr
      double precision z(n1,n2,n3)
      integer m1, m2, m3

      m1 = min(n1,18)
      m2 = min(n2,14)
      m3 = min(n3,18)

      write(*,*)'  '
      do  i=0,nprocs-1
         if( me .eq. i )then
            write(*,*)' id = ', me
            do  i3=1,m3
               do  i1=1,m1
                  write(*,6)(z(i1,i2,i3),i2=1,m2)
               enddo
               write(*,*)' - - - - - - - '
            enddo
            write(*,*)'  '
 6          format(15f6.3)
         endif
         call mpi_barrier(mpi_comm_world,ierr)
      enddo

      return
      end

      subroutine zero3(z,n1,n2,n3)

      implicit none

      include 'mpinpb.h'

      integer n1, n2, n3
      double precision z(n1,n2,n3)
      integer i1, i2, i3

      do  i3=1,n3
         do  i2=1,n2
            do  i1=1,n1
               z(i1,i2,i3)=0.0D0
            enddo
         enddo
      enddo

      return
      end
bradc-lnx%
```

2) *rprj3* Stencil from NAS MG in Chapel

```
def rprj3(S: [?SD], R: [?RD]) {
  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 ijk in SD do
    S[ijk] = + reduce [offset in Stencil]
                                 (W3D[offset] * R[ijk + RD.stride*offset]);
}
```



Our previous work in ZPL demonstrated that such compact codes can result in better performance than Fortran + MPI while also supporting more flexibility at runtime*.

****e.g.*, the Fortran + MPI *rprj3* code shown previously not only assumes *p* divides *n*, it also assumes that *p* and *n* are specified at compile-time and powers of two.**

2) Classifying Current Programming Models

	System	Data Model	Control Model
Communication Libraries	MPI/MPI-2	Local-View	Local-View
	SHMEM, ARMCI, GASNet	Local-View	SPMD
Shared Memory	OpenMP, Pthreads	Global-View (trivially)	Global-View (trivially)
PGAS Languages	Co-Array Fortran	Local-View	SPMD
	UPC	Global-View	SPMD
	Titanium	Local-View	SPMD
PGAS Libraries	Global Arrays	Global-View	SPMD

2) Classifying Current Programming Models

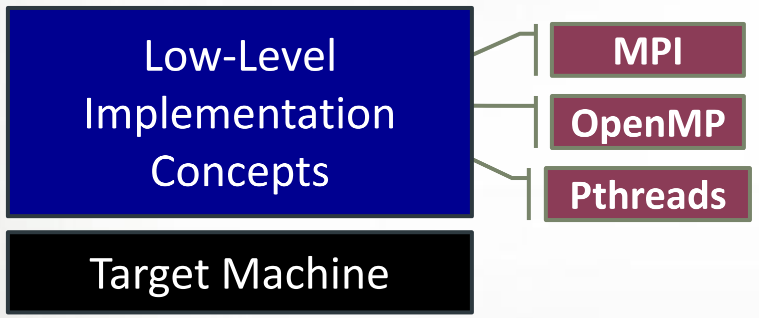
	System	Data Model	Control Model
Communication Libraries	MPI/MPI-2	Local-View	Local-View
	SHMEM, ARMCI, GASNet	Local-View	SPMD
Shared Memory	OpenMP, Pthreads	Global-View (trivially)	Global-View (trivially)
PGAS Languages	Co-Array Fortran	Local-View	SPMD
	UPC	Global-View	SPMD
	Titanium	Local-View	SPMD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

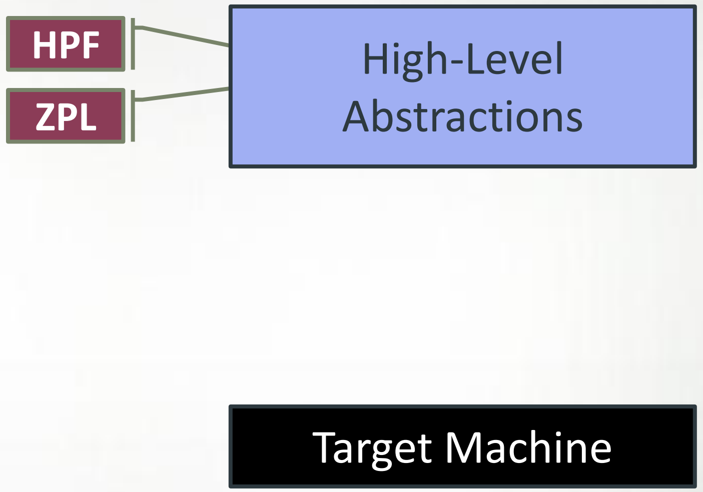
2) Global-View Programming: A Final Note

- A language may support both global- and local-view programming — in particular, Chapel does

```
def main() {
  coforall loc in Locales do
    on loc do
      MySPMDProgram(loc.id, Locales.numElements);
}

def MySPMDProgram(me, p) {
  ...
}
```

3) Multiresolution Language Design: Motivation





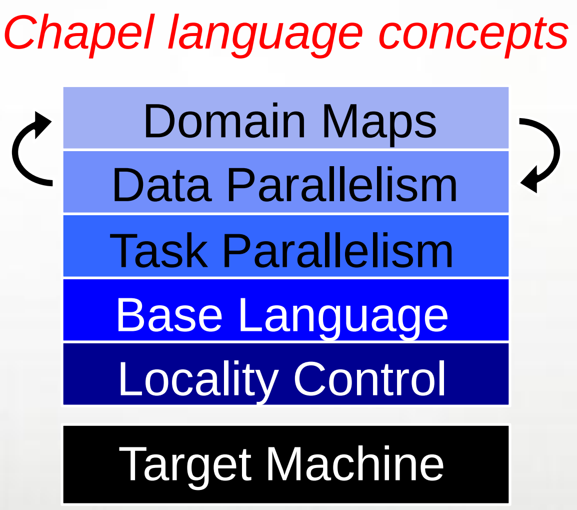
"Why is everything so difficult?"
"Why don't my programs port trivially?"

"Why don't I have more control?"

3) Multiresolution Language Design

Multiresolution Design: Support multiple tiers of features

- higher levels for programmability, productivity
- lower levels for performance, control
- build the higher-level concepts in terms of the lower



- separate concerns appropriately for clean design

4) Control over Locality/Affinity

Consider:

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

Therefore:

- Placement of data relative to computation affects scalability
- Programmers need control over data and task placement

Note:

- As core counts grow, locality will matter more on desktops
- GPUs and accelerators already expose node-level locality

5) Reduce Gap Between HPC & Mainstream Languages

Consider:

- Students graduate with training in Java, Matlab, Perl, Python
- Yet HPC programming is dominated by Fortran, C/C++, MPI

We'd like to narrow this gulf with Chapel:

- to leverage advances in modern language design
- to better utilize the skills of the entry-level workforce...
- ...while not ostracizing the traditional HPC programmer
  - e.g., support object-oriented programming, but make it optional

Other examples:

- function overloading, name-based argument passing
- scripting-like features: type inference, generic functions
- rich data structures with iterators (*e.g.*, associative arrays)

Questions?

- Chapel's Context
- Chapel's Motivating Themes
  1. General parallel programming
  2. *Global-view* abstractions
  3. *Multiresolution* design
  4. Control over locality/affinity
  5. Reduce gap between mainstream & HPC languages